AD-A216 791

DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited. (2)
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 89-1880
6a. NAME OF PERFORMING ORGANIZATION Johns Hopkins University	6b. OFFICE SYMBOL (If applicable)
6c. ADDRESS (City, State, and ZIP Code) Applied Physics Laboratory Laurel, MD 20707-6099	7a. NAME OF MONITORING ORGANIZATION Air Force Office of Scientific Research 7b. ADDRESS (City, State, and ZIP Code) Building 410 Bolling AFB, DC 20332-6448
8a. NAME OF FUNDING / SPONSORING ORGANIZATION AFOSR	8b. OFFICE SYMBOL (If applicable) NM
8c. ADDRESS (City, State, and ZIP Code) Building 410 Bolling AFB, DC 20332-6448	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER AFOSR-87-0354
	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO. A7; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

INVESTIGATION OF NEURAL NETWORK DYNAMICS

12. PERSONAL AUTHOR(S)

Dr. Fernando J. Pineda

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
FINAL	FROM Sep 87 TO 31 Jan89		

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The purpose of this document is to report the progress made in the first year of the grant entitled "Investigation of Neural Network Dynamics" (AFOSR-87-0354). The proposed period of the work was September 1, 1987 to August 31, 1990. The proposed three year budget was $126,200 with a first year budget of $40,000. The grant was closed after a single year because the principal investigator moved from the Applied Physics Laboratory, Johns Hopkins University to the Jet Propulsion Laboratory, California Institute of Technology. Nevertheless, many of the initial objectives were met in the single year that the grant was in force. The major result of this investigation is a systematic approach for exploiting the dynamics of a general class of neurodynamical systems for the purpose of neural computation. We have interpreted the back-propagation formalism as an

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL DR. ABRAHAM WAKSMAN	22b. TELEPHONE (Include Area Code) (202) 767-5027
	22c. OFFICE SYMBOL NM

DD Form 1473, JUN 86 — Previous editions are obsolete. — SECURITY CLASSIFICATION OF THIS PAGE

adaptive algorithm for a general class of dynamical systems. The completely continuous formalism lends itself to implementation in analog VLSI. This can be accomplished without external synchronization. Four papers were published which acknowledged the grant. Three were published in refereed Journals, the third in a refereed conference. One student was partially supported by the grant. The work received wide recognition and acceptance from the scientific and technical community.

FINAL REPORT:

INVESTIGATION OF NEURAL NETWORK DYNAMICS

(AFOSR-87-0354)

Aug. 28,1989

Fernando J. Pineda (P.I.)

formerly at:
Applied Physics Laboratory, Johns Hopkins University
Laurel MD, 20707

presently at:
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Dr., Pasadena CA, 91109

Abstract

The purpose of this document is to report the progress made in the first year of the grant entitled "Investigation of Neural Network Dynamics" (AFOSR-87-0354). The proposed period of the work was September 1, 1987 to August 31, 1990. The proposed three year budget was $126,200 with a first year budget of $40,000.

The grant was closed after a single year because the principal investigator moved from the Applied Physics Laboratory, Johns Hopkins University to the Jet Propulsion Laboratory, California Institute of Technology. Nevertheless, many of the initial objectives were met in the single year that the grant was in force.

The major result of this investigation is a systematic approach for exploiting the dynamics of a general class of neurodynamical systems for the purpose of neural computation. We have interpreted the back-propagation formalism as an adaptive algorithm for a general class of dynamical systems. The completely continuous formalism lends itself to implementation in analog VLSI. This can be accomplished without external synchronization.

Four papers were published which acknowledged the grant. Three were published in refereed Journals, the third in a refereed conference. One student was partially supported by the grant. The work received wide recognition and acceptance from the scientific and technical community.

1. Introduction

This document reports on progress made in the first year of the grant entitled "Investigation of Neural Network Dynamics" (AFOSR-87-0354). The proposed period of the work was September 1, 1987 to August 31, 1990. The proposed three year budget was $126,200 with a first year budget of $40,000. The term of the first year grant was extended until January 31, 1989.

2.1 Research objectives/Statement of work

To achieve a substantial improvement in the understanding of neural network dynamics a program based on numerical simulation and formal analysis was proposed. The investigation was based on several broad issues which were address:

1) What are the formal relationships between the various neural network models? Models were expected to fall into equivalence classes which displayed qualitatively similar behavior. It was of interest to identify the minimal models in each class and to identify the most general models in each class.

2) It was proposed to investigate the conjecture that the convergence of some networks was an emergent property -- that is, the convergence of the system improved in the limit of very many processing units.

3) It was proposed to investigate the storage capacity of various neural networks. A detailed analysis of the information storage capacity of feedforward networks had been performed by Baum et al. [BMW87]. We were interested in extending these results to networks with feedback.

4) It was proposed to investigate how particular models map onto particular machines and what models lend themselves to implementation in VLSI. Ideally, a model suitable for implementation in VLSI would keep a maximal amount of silicon busy. This means that as many units as possible must take an active role in the computation. A desirable property is for each node to be able to process its information asynchronously. We were interested in investigating models which possess this property.

5) It was proposed to investigate the role of characteristic time scales in neural networks. There are at least three time scales which play a role in neural networks. The existance of three time scales can be inferred: 1) signal propagation time, τ_P, in the brain this corresponds to the 1-10 mS which is the time it takes signals to propagate across the cortex; 2) time of input fluctuations, τ_I, the brain responds to inputs from the external environment which fluctuate with their own characteristic time scales; 3) time scale of microscopic learning, τ_L, in the brain the strength of a synaptic connection changes with time.

6) It was proposed to investigate how algorithms scale in the limit of many processors (large-N limit). An important field of research at the present time is the development of faster learning algorithms which can be scaled up to networks with millions of modifiable connections.

2.2 Personnel

The investigations were carried out by the following personnel: The principal investigator for this program was Dr. Fernando J. Pineda. During the first year of the grant he was a member of the Senior Professional Staff in the Space Department at JHU/APL. Currently he is a member of the Technical Staff at the Jet propulsion Laboratory, California Institute of Technology. Dr. Pineda received his Ph.D. in Theoretical Nuclear Physics from the University of Maryland in December 1986. Part-time support was provided for a very talented undergraduate student from Johns Hopkins University: Mr. A. David Redish. In addition, the P.I. worked with two students: Mr. Ben Yuhas of the Deptartment of Computer and Electrical Engineering at Johns Hopkins University, and Mr. Etienne Duprit of the Naval Research Laboratories. Mr. Duprit published his class project in the refereed journal Neural Networks[De89].

2.3 Status of work

This section addresses the progress made in the first year. Most of the progress was made in the area of adaptive algorithms based on the Recurrent Backpropagation (RBP) algorithm. The progress made during the first year was reported in the following three papers:

1) Generalization of back-propagation to recurrent neural networks,
F. J. Pineda, *Physical Review Letters*, **18**, pp. 2229-2232, (1987)

2) Generalization of back-propagation to recurrent and higher-order neural networks,
F. J. Pineda, (to appear), Proceedings of IEEE Conference on Neural Information Processing Systems, (Dana Z. Anderson, ed.), Denver Colorado, Nov. 8-12, (1987)

3) Dynamics and Architecture in neural computation,
F. J. Pineda, (to appear), *Journal of Complexity*, Special Issue on Neural Networks, September , (1988)

In addition to the above, some work was reported at the "Neural Networks for Computing" conference held in Snowbird Utah, April 6-10, (1988). Finally, after the grant had expired, the author wrote a mini-review article entitled "Recurrent Backpropagation and the Dynamical Approach to Neural Computation" [Pi89]. It summarized the work performed to date and its significance. It is included in the appendix.

Some of the key results of the investigation are now summarized. The reader may refer to the articles in the appendices for detailed information.

1) Our investigtion into adapative algorithms was based on a formalism for constructing general adaptive dynamical systems which obey nonlinear coupled differential equations. This formalism, denoted Recurrent back-propagation (RBP) is a non-algorithmic continuous-time formalism for adaptive recurrent and nonrecurrent networks in which the physical aspects of the computation are stressed [Pi87a, Pi87b and Pi88]. The formalism is expressed in the language of differential equations so that the connection to collective physical systems is more natural. RBP can be put into an algorithmic form to optimize the performance of the network on digital machines, nevertheless, as shall be discussed below, the intent of the formalism is to stay as close to collective physics and dynamics as possible.

The class of neural network models which can be trained by RBP is very general, but most of our work has focused on a simple system given by the following differential equations:

$$\tau_x dx_i/dt = -x_i + \sum_j w_{ij} f(x_j) + I_i \qquad i = 1, \ldots, n \ . \qquad (1)$$

The vector **x** represents the state vector of the network, **I** represents an external input vector and **w** represents a matrix of coupling constants (weights) which represent the strengths of the interactions between the various neurons. The relaxation time scale is τ_x. By hypothesis, the vector valued function $f(x_i)$ is differentiable and chosen so as to give the system appropriate dynamical properties. For example, biologically motivated choices are the logistic or hyperbolic tangent functions [Co68]. When the matrix **w** is symmetric this system corresponds to the Hopfield model with graded neurons [Ho84].

In general , the solutions of equation (1) exhibit oscillations, convergence onto isolated fixed points and chaos. For our purposes, convergence onto isolated fixed points is the desired behavior, because we use the value of the fixed point as the output of the system. When the network is loaded,

the weights are adjusted so that the output of the network is the desired output. Recurrent Back-propagation dynamics is based on gradient descent and exploits two tricks to reduce the amount of computation. The first trick uses the fact that, for equations of the form (1), the gradient of an objective function $E(x^o)$ can be written as an outer-product, i.e.

$$\nabla_w E = y^o f(x^o)^T \quad . \tag{2}$$

Where x^o is the fixed point of eqn. (1) and where the "error vector" y^o is given by

$$y^o = (L^T)^{-1} J \tag{3}$$

where L^T is the transpose of the matrix n x n matrix whose components are

$$L_{ij} = \delta_{ij} - w_{ij} f'(x_j) .$$

J is an external error signal which depends on the objective function and on x^o. This trick reduces the computational complexity of the gradient calculation by a factor of n because L^{-1} can be calculated by direct matrix inversion in $O(n^3)$ operations and because x^o can be calculated in only $O(n^2)$ calculations. Thus the entire calculation scales like $O(n^3)$ or $O(N^{3/2})$. The second trick exploits the fact that y^o can be calculated by relaxation or equivalently it is the (stable) fixed point of the following coupled set of linear differential equation:

$$\tau_y dy_i/dt = -y_i + f'(x_i)\sum_j w_{ji} y_j + J_i \tag{4}$$

This equation was derived by [Pi87]. A discrete-time version was derived independently by Almeida [Al87].

2) The recurrent backpropagation formalism provides a single unified approach for feed-forward type networks and recurrent networks. This approach is very powerful and can be applied to a variety of dynamical systems. Another consequence of the unified approach is that it is possible to build heirarchical architectures containing both associative memory and feed-forward components from homogenous units. This was demonstrated in [Pi88].

3) The role of characteristic time scales in the adaptive algorithms was investigated. In many paradigms of neural computation (e.g. conventional backpropagation) characteristic time scales do not play a role. Therefore it is impossible to gauge how fast a "true" neural machine would perform the same task. (For this purpose a "true" neural machine is one whose basic functional units implement the appropriate neurodynamics directly --as typified by the analog VLSI approach taken by Mead [Me89]. Constraints on various time scales were derived and published [Pi88].

4) We have shown how the algorithm can be implemented as a continous-time dynamical system. This is important because it permits the possibility of implementing the dynamical learning algorithms in analog VLSI without the need for an internal or external clock. Our formalism has allowed us to estimate the performance of analog VLSI implementations of backpropagation networks. Preliminary results were presented at the Snowbird conference [Pi88]. The performance of an electronic physical system was estimated by using electronic time scales for the characteristic time scales. For a simple example, it was estimated that learning could be accomplished in approximately 10 milliseconds. This compared very well to a digital simulation that required several minutes to converge.

5) We proposed to investigate how particular models map onto particular machines and what models lend themselves to implementation in VLSI. Mr. Etienne Duprit, a student taking a course taught by the P.I. implemented the recurrent backpropagation algorithm on a Connection Machine [De88]. Mr. Duprit, experimented with two different implementations of the algorithm. One implementation was based on a generalization of the implementation of Rosenberg and Blelloch [RB88]. This implementation devoted CM processors to connections as well as neurons. The second implementation was based on a routing algorithm developed by Tomboulian [To86] which used processors for neurons only. Mr. Deprit concluded that the Tomboulian algorithm spends most of it's time communicating whereas the Rosenberg and Blelloch algorithm spends most of its time computing. Furthermore, for networks where the fan-in gets very large the the Tombolian algorithm required processors with memory size proportional to fan-in. The most effective use is made of hardware if the connections themselves can be made to perform computation rather than the processors. In otherwords synapses must be "smart" memories.

6) We Investigated how the gradient calculation algorithm scales with the number of processors. In gradient descent learning, the computational problem is to optimize an objective function whose free parameters are the weights. Let the number of weights be denoted by "N" and let the number of processing units be denoted by "n". Then, N is proportional to n^2 if the fan-in/fan-out of the units is proportional to n. In a neural network the evaluation of an objective function requires O(N) or $O(n^2)$ operations. Accordingly, to calculate the gradient of the objective function by numerical differentiation requires $O(N^2)$ or $O(n^4)$ calculations. For big problems, i.e. problems with lots of connections this becomes intractable very rapidly.

To relax $\mathbf{y}$ (i.e. to integrate eqn. (4) until $\mathbf{y}$ reaches steady state) requires $O(n^2)$ operations per time step. The number of time steps is independent of n. Therefore the calculation of $\mathbf{y}^o$ is $O(n^2)$ or O(N). The method is computationally efficient provided the network is sufficiently large and sparse

and provided that the fixed points are not marginally stable. These results are summarized in table 1. Note that the two back-propagation algorithms scale like O(N), but this hides the constants of proportionality which for feed-forward networks depends on the number of layers where as for recurrent networks the constant of proportionality depends strongly on the eigenvalues of the **L** matrix. Indeed, if the fixed points are marginally stable, the number of iterations required to converge onto x^o and y^o may diverge. The relationship between various algorithms is shown in the table below

numerical algorithm	**complexity**
Worst case (e.g. numerical differentiation)	$O(N^2)$
matrix inversion (e.g. gaussian elimination)	$O(N^{3/2})$
matrix inversion by relaxation (e.g. RBP)	O(N)
recursion (e.g. classical feed-forward back-propagation)	O(N)

Table 1. Scaling of gradient calculation with the number of connections

The scaling referred to here should not be confused with the number of gradient evaluations required for convergence to a solution. Indeed, for some problems, e.g. parity, the required number of gradient evaluations may diverge at critical training set sizes [Te87].

The O(N) scaling of the gradient calculation is arguably the single most important reason that back-propagation algorithms have made such an impact. The idea of using gradient descent is certainly not new, but whereas it was previously tractable on small problems only, it is now tractable on big problems to. It is interesting to observe that a similar situation arose after the development of the FFT algorithm. The idea of numerical fourier transforms had been around for a long time before the FFT, but the FFT caused a computational revolution by reducing the complexity of an n-point fourier transform, from $O(n^2)$ to $O(n \cdot \log(n))$.

3. Related funding actions

3.1 AFOSR

A proposal to continue the work in this grant has been submitted to the AFOSR through the Jet Propulsion Laboratory, California Insititute of Technology. The new proposal is entitled: "Adaptive Dynamics for Neural Computation.", JPL Task plan No. 80-3095

3.2 Recognition of work by the community

Recurrent Back-propagation has proven to be a rich and useful computational tool. Qian and Sejnowski [QS88] have demonstrated that a recurrent back-propagation network can be trained to calculate stereo disparity. This results in a network similar to that of Marr and Poggio [MP76]. Barhen et al. [BGZ89] have used the method to train networks on inverse kinematics for robotic applications. The formalism has also been fertile soil for theoretical developements. Pearlmutter [Pe88] has extended the technique to time dependent trajectories while Simard and Ballard [SB88] have investigated its convergence properties.

The principal investigator has been invited to present talks based on this work at the University of Chicago, (Depts.of Mathematics and Computer Science), The California Institute of Technology, (Dept. of Electrical Engineering). Prof. Carver Mead at Caltech has expressed an interest in applying these ideas to the construction of an actual analog VLSI learning chip. Most recently, the P.I. has been invited to contribute a chapter to a book devoted to the backpropagation algorithm (edited by D. Rumelhart and E. Chauvin).

6. References

Al87 Almeida, L. B. (1987). A Learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In: *Proceedings of the IEEE First International Converence on Neural Networks*, (eds. M. Caudil and C. Butler), Vol. 2., pp. 609-618, San Diego CA

BGZ89 Barhen, J., Gulati, S., Zak, M.(1989). Neural Learning of Constrained Nonlinear Transformation, 22, 67-76

BMW87 Eric B. Baum, John Moody and Frank Wilczek, *Internal Representations for Associative Memory*, ITP reprint, Institute for Theoretical Physics, University of California Santa Barbara, CA (1987)

De89 Deprit, E. (1989). Implementing Recurrent Backpropagation on the Connection Machine, *Neural Networks*, 2, 295-314

Ho84 J.J. Hopfield, *Neurons with graded response have collective computational properties like those of two-state neurons*, Proc. Nat. Acad. Sci. USA, Bio. 81,3088-3092, (1984)

Pe89 Pearlmutter,B.A., (1989), Learning State Space Trajectories in Recurrent Neural Networks, *Neural Computation*

Pi87a Pineda, F. J. (1987a). Generalization of back-propagation to recurrent neural networks. *Phys. Rev. Lett.* 18, pp. 2229-2232

Pi87b Pineda, F. J. (1987b). Generalization of back-propagation to recurrent and higher order networks. In: *Proceedings of IEEE Conference on Neural Information Processing Systems* (ed. D. Z. Anderson), Denver Colorado, Nov. 8-12

Pi88 Pineda, F. J. (1988). Dynamics and Architecture for Neural Computation. *Journal of Complexity*. 4, pp. 216-245

Pi89 Pineda, F.J.,(1989). Recurrent-Backpropagation and the Dynamical approach to Adaptive Neural Computation, *Neural Computation*, 1, 161-172

MP76 Marr, D. and Poggio, T. (1976). Cooperative Computation of Stereo Disparity, *Science*, 194, 283-287

Me89 Mead, C. (1989), "Analog VLSI and Neural Systems", Addison-Wesley, Reading MA

Ro86 Rosenberg, C.R. and Blelloch, G.E., (1986). An Implementation of Network Learning on the Connection Machine, Technical Report, Thinking Machines Corporation, Cambridge, MA

RHW86 D. E. Rumelhart, G. E. Hinton and R.J. Williams, *Learning Internal Representations by Error Propagation*, in Parallel Distributed Processing, eds. D.E. Rumelhart and J.L. McClelland, M.I.T. press, (1986)

SB88 Simard, P.Y., Ottaway, M. B. and Ballard, Dana, H. (1988). Analysis of Recurrent Backpropagation, *Proceedings of the 1988 Connectionist Models Summer School*, June 17-26, 1988, Carnegie Mellon Univ., Morgan Kaufman Publishers

Te87 Tesauro, Gerald, (1987). Scaling relationships in back-propagation learning: dependence on training set size, *Complex Systems*, 1, pp. 367-372

To88 Tomboulian, S.J., (1988) A Brief Overview of a System for Routing Directed Graphs on SIMD Architectures, *Proceedings of 2nd* Symposium on Frontiers of Massively Parallel Computation, Fairfax, VA (1988)

Appendices

A.1 **Generalization of Back-Propagation to Recurrent Neural Networks** Physical Review Letters, 59, pp.2229-2232, (1987)

A.2 **Generalization of Back-Propagation to Recurrent and Higher Order Neural Networks**, Neural Information Processing Systems, (Dana Z. Anderson, ed.) American Institute of Physics, New York, (1988)

A.3 **Dynamics and Architecture for Neural Computation**, Journal of Complexity 4, 216-245, (1988)

A.4 **Recurrent Back-Propagation and the Dynamical Approach to Adaptive Neural Computation**, Neural Computation, 1, 161-172, (1989)

Generalization of Back-Propagation to Recurrent Neural Networks

Fernando J. Pineda

Applied Physics Laboratory, Johns Hopkins University, Laurel, Maryland 20707

(Received 10 June 1987)

An adaptive neural network with asymmetric connections is introduced. This network is related to the Hopfield network with graded neurons and uses a recurrent generalization of the δ rule of Rumelhart, Hinton, and Williams to modify adaptively the synaptic weights. The new network bears a resemblance to the master/slave network of Lapedes and Farber, but it is architecturally simpler.

PACS numbers: 87.30.Gy

The neural network approach is a paradigm for computation in which the traditional paradigm of a finite-state machine performing sequential instructions in a discrete state space is replaced with the paradigm of a dynamical system, in a discrete or continuous state space, which evolves under the control of a certain class of dynamics (*neurodynamics*). Although a precise definition of neurodynamics does not exist, it seems safe to characterize it by at least three salient features. First, the dynamical system has very many degrees of freedom. At the present time, most simulations of these systems are limited to less than 10^5 neurons. On the other hand, the human brain has at least 10^{11} neurons. The activity level and the time derivative of the activity of the neurons are the coordinates in the phase space of the system. This phase space plays the role of the state space in a conventional computing machine. The second feature of neurodynamics is nonlinearity. Nonlinearity is essential to create a universal computing machine. This follows because a network composed of linear units can always be reduced to an equivalent single-layer network which performs the same input/output transformation. But, as pointed out by Minsky and Papert,[1] a universal computing machine cannot be built from a single layer of finite-order neurons. The third feature of neurodynamics is that it is dissipative. A dissipative system is characterized by the convergence of the phase-space volume onto a manifold of lower dimensionality as time increases. Systems whose flow exhibits the property of *global asymptotic stability* play a particularly important role in neural-network modeling. Global asymptotic stability implies that the system will ultimately settle down to a steady state for any choice of initial condition. Systems which minimize an energy function, such as the Hopfield model, are guaranteed to be globally asymptotically stable.

The identification of stable fixed points with computational objects, e.g., memories, is one of the fundamental ideas of the paradigm. To implement this idea it is necessary to control the locations of the fixed points of the neural networks. A learning algorithm is a rule or dynamical equation which changes the locations of fixed points to encode information. One way of doing this is to minimize, by gradient descent, some function of the system parameters. This general approach is reviewed by Amari[2] and forms the basis of many learning algorithms. The algorithm described here is a specific case of this general approach.

The dynamics of the network considered in this Letter is based on the following system of coupled differential equations:

$$dx_i/dt = -\alpha x_i + \beta f_i\left(\sum_j w_{ij}x_j\right) + I_i, \tag{1}$$

where x_i represents the activity of the ith neuron, where the matrix element w_{ij} denotes the connection strength, or coupling, from the jth to the ith neuron, and where α and β are conveniently chosen positive constants. The functions f_i may have different forms for various populations of neurons. A commonly used form is the logistic function,

$$f(\xi) = (1+e^{-\xi})^{-1}.$$

The constant I_i represents an external input bias which may be included inside or outside $f(\xi)$. I chose the latter case arbitrarily. The fixed points of (1), which I denote as $\mathbf{x}^0$, are solutions of the nonlinear algebraic equations

$$\alpha x_i^0 = \beta f_i\left(\sum_j w_{ij}x_j^0\right) + I_i \tag{2}$$

and are implicit functions of the weight matrix $\mathbf{w}$ and initial state $\mathbf{x}^i$.

Suppose that $\mathbf{w}$ is lower triangular. Then it is clear that Eq. (2) can be solved recursively since to calculate x_i one needs only $x_1, \ldots, x_{i-1}$. Thus, when the units are properly labeled, this is just the forward propagation which occurs in the widely used feedforward network of Rumelhart, Hinton, and Williams.[3] I conclude that the feedforward network simply provides a direct method of calculating the fixed points of (1) when $\mathbf{w}$ is lower triangular.

The δ rule is a learning rule for feedforward networks. Strictly speaking, it is restricted to feedforward networks only. Nevertheless it has been applied to recurrent networks by taking advantage of the fact that for every recurrent network there exists an equivalent feedforward network (for a finite time). The cost for this strategy is

the manyfold duplication of the hardware for the feedforward version of the recurrent network.[3] The algorithm presented in this paper makes unnecessary the artifice of unfolding a recurrent network into a feedforward network.

A necessary condition for the learning algorithm discussed in this Letter to exist is that system (1) reach steady state (I will not discuss limit cycles here). Except for some theorems concerning collective quantities,[4] little is known about the stability of system (1) for arbitrary $\mathbf{w}$. However, there are special cases for which (1) can be proved to be globally asymptotically stable. The set of equations (1) is stable if $\mathbf{w}$ is symmetric because (1) can be transformed into the equations studied by Hopfield[5] under the coordinate transformation,

$$u_i = \sum_k w_{ik} x_k.$$

Hopfield's equations are globally asymptotically stable if $\mathbf{w}$ is symmetric and has zeros along the diagonal. Stability in this case is proved because there exists a Liapunov function. A general theorem concerning stability of networks with symmetric weights is given by Cohen and Grossberg.[6] The set of equations (1) is also globally asymptotically stable if $\mathbf{w}$ is lower triangular because in such a case the network is a pure feedforward network. In other words the nth unit can only receive input from the mth unit if $n > m$. The stability of the feedforward case follows from a recursive agreement which goes as follows. Suppose that the activations x_i (where $i = 1, \ldots, m$) are constant. Then from the feedforward constraint the nth unit (where $n = m + 1$) receives only constant input. With constant input Eqs. (1) converge exponentially to a constant value, and hence x_{m+1} becomes constant. Thus it is clear that if the inputs are constant, the activation of the entire network will ultimately become constant. Equations (1) are also stable in the limit of infinite $\mathbf{w}$ since if $\mathbf{w}$ is infinite the function $f(u_i)$ becomes constant and the solutions to (1) simply decay exponentially to constants.

Numerical simulations conducted by this author strongly suggest that in practice the system is stable for most $\mathbf{w}$ and initial $\mathbf{x}$. Oscillatory solutions can occur when there exists substantial self-excitation. It shall be assumed, for the purpose of deriving the back-propagation equations, that the system ultimately settles down to a stable state. With this caveat in mind I present the recurrent back-propagation (RBP) algorithm.

Consider a system of neurons, or units, whose dynamics is determined by Eqs. (1). Of all the units in the network we will arbitrarily define some subset of them, A, as *input* units and some other subset of them, Ω, as *output* units. Units which are neither members of A or Ω are denoted *hidden* units. A unit may be simultaneously an input unit and an output unit. If a unit is an input unit, the corresponding component of the vector $\mathbf{I}$ is nonzero and represents an external input to the system, i.e.,

$$I_i = \begin{cases} \xi_i, & \text{if } i \in A, \\ 0, & \text{otherwise,} \end{cases}$$

where ξ_i is an external input.

Our goal will be to find a local algorithm which adjusts $\mathbf{w}$ so that a given fixed initial state $\mathbf{x}^i$ and a given set of input values ξ_i result in a fixed point, $\mathbf{x}^0$, whose components along the output units have a desired set of values, τ_j (where $j \in \Omega$). This will be accomplished by our minimizing a function which measures the error between the desired fixed point and the actual fixed point. Consider the positive definite function

$$E(\mathbf{x}^0) = \tfrac{1}{2} \sum_{i=1}^{N} J_i^2,$$

where

$$J_i = \begin{cases} \tau_i - x_i^0, & \text{if } i \in \Omega, \\ 0, & \text{otherwise.} \end{cases}$$

It is an implicit function of the weight matrix $\mathbf{w}$ because the fixed point $\mathbf{x}^0$ is implicitly dependent on the weight matrix. $E(\mathbf{x}^0)$ has a family of minima which exist on the hyperplanes which satisfy $x_i^0 = \tau_i$, where $i \in \Omega$.

A formal learning algorithm consists of an algorithm which drives the fixed point towards one of these hyperplanes. Dynamically, this is accomplished by our letting the system evolve in the weight space along trajectories which are antiparallel to the gradient $\partial E/\partial w_{ij}$. In other words,

$$dw_{ij}/dt = -\eta\, \partial E/\partial w_{ij}, \tag{3}$$

where η is a numerical constant which defines the (slow) time scale on which $\mathbf{w}$ changes. η *must be small so that* x *is always essentially at steady state* (i.e., $\mathbf{x} \cong \mathbf{x}^0$). On performing the differentiations in (3) one immediately obtains

$$dw_{rs}/dt = \eta \sum_k J_k\, \partial x_k^0/\partial w_{rs}. \tag{4}$$

The derivative of x_k^0 with respect to w_{rs} is obtained by our differentiating both sides of (2) with respect to w_{rs} and solving for the derivatives. The result is

$$\partial x_k^0/\partial w_{rs} = \beta(\mathbf{L}^{-1})_{kr} f_r'(u_r) x_s^0, \tag{5}$$

where the matrix $\mathbf{L}$ is given by

$$L_{ij} = \alpha\delta_{ij} - \beta f_i'(u_i) w_{ij},$$

and where δ_{ij} is the Kronecker δ symbol. On substituting (5) into (4) one immediately obtains

$$dw_{rs}/dt = \eta y_r x_s^0, \tag{6}$$

where

$$y_r = \beta f_r'(u_r) \sum_k J_k (\mathbf{L}^{-1})_{kr}. \tag{7}$$

Equations (6) and (7) specify a formal learning rule. Equations (7) require a matrix inversion to calculate the error signals, y_k. Direct matrix inversions are necessarily nonlocal calculations and therefore this learning algorithm is not suitable for implementation as a neural network. A local method for the calculation of y_r is obtained by the introduction of an associated dynamical system. Consider the vector z whose components are defined in terms of the components of y according to

$$y_r = \beta f_r'(u_r) z_r. \tag{8}$$

Equations (7) and (8) imply that z_r satisfies

$$\sum_r L_{ri} z_r = J_i. \tag{9}$$

Now observe that the solutions of Eqs. (9) are the steady-state solutions of

$$dz_i/dt = -\sum_r L_{ri} z_r + J_i.$$

In terms of the explicit variables in the problem, these equations are

$$dz_i/dt = -\alpha z_i + \beta \sum_r \{f_r'(u_r) w_{ri} z_r\} + J_i. \tag{10}$$

This leads to a learning rule of the form

$$dw_{rs}/dt = \eta f'(u_r) z_r^0 x_s^0. \tag{11}$$

Equations (1), (10), and (11) completely specify the dynamics for an adaptive neural network, provided that (1) and (10) are convergent. It is known that the convergence of (1) is a sufficient condition for the convergence of (10).[7] This follows from the observation that the back-propagation network is obtained from the forward-propagation network (linearized about a fixed point) and that a linear network is stable in both directions if it is stable in either direction. It is quite easy for one to obtain the δ rule from the RBP algorithm by expressing Eqs. (1), (10), and (11) as difference equations with $\Delta t = 1$ and with w lower triangular.

I have conducted preliminary numerical experiments with exclusive OR (XOR) networks to verify the correctness of the algorithm. These were performed by my approximating the differential equations with first-order finite-difference equations and requiring that Eqs. (1) and (10) converge before taking an integration step in Eq. (11). The XOR network is shown in Fig. 1. Each input unit receives one digit of a two-digit binary number. The target x_i for the output unit is 1 if the number of 1's in the input is odd and 0 otherwise. Unit 5 is a threshold unit, i.e., it biases the total input to units 3 and 4 so as to provide a threshold which must be exceeded if these units are to turn on. Unit 5 feeds back on itself so as to stay turned on always. The feedforward exclusive OR network used by Rumelhart, Hinton, and Williams is completely equivalent to this network if the backward connection from unit 4 to unit 3 is omitted and if the feedback loop in unit 5 has an infinite positive magnitude. In practice I made the magnitude of the loop merely large and was able to reproduce the behavior of the Rumelhart network.

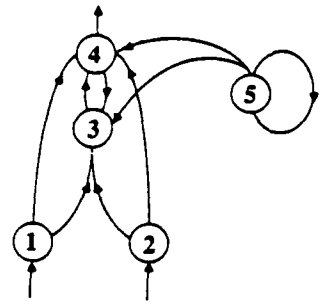

FIG. 1. XOR network with recurrent connections.

The network with the backward connection performed only modestly faster than the network without this connection.[8] The main difference in the networks was in the distribution of final weights. Both networks had similar attractors which could be characterized by the final root mean square weight per connection (w_{rms}). The attractor which I denote by A had $w_{rms} = 3.4$, while the attractor denoted by B had $w_{rms} = 10.7$. The network without the backward connection converged onto attractors A and B approximately 85% and 15% of the trials, respectively, whereas the network with the backward connection converged onto attractors A and B approximately 52% and 48% of the trials, respectively. Only in one trial out of 480 did the recurrent network fail to converge onto a global minimum. Each pattern was presented to the recurrent network approximately 200 times. The final solutions were insensitive to the initial value of x which indicates that the attractors of Eqs. (1) have large basins of convergence.

It is worthwhile to compare the RBP network with the master/slave network of Lapedes and Farber.[9] The slave network corresponds to my forward-propagation network. If we suppose it has N nodes then the master network determines the weights of the slave network by integrating N^2 equations, each of which has a form similar to Eqs. (1), but with slave weight matrix elements as dynamical variables and a rank-4 matrix as the master's weight matrix. The weight matrix of the master network has a simple symmetric form with at most $N(N+1)/2$ nonvanishing independent components. These components require additional storage beyond the N^2 components of the slave's weight matrix. The RBP network, on the other hand, requires the integration of N^2+2N equations and no additional storage. $2N$ of these equations correspond to Eqs. (1) and (10). The remaining N^2 equations have a simple outer product form [cf. Eq. (11)] and are quite trivial to implement. The conclusion is that the RBP network is an architecturally simpler network than the master/slave network and requires less memory.

The master/slave network directly minimizes the average of E over all input/output associations. This average is denoted by $\langle E \rangle$. The master equations are guaranteed to converge to at least a local minimum of $\langle E \rangle$ because $\langle E \rangle$ is a Liapunov function for the equations.[10] On the other hand, E is a Liapunov function for Eq. (3) of the RBP network only in the case of a single input/output association. For multiple associations the RBP network is guaranteed to converge only in a probabilistic sense and under certain technical conditions. It was noted by Amari[2] that gradient-descent algorithms, such as RBP, converge to a minimum point of $\langle E \rangle$ to within a small fluctuating term provided that the input/output sequence is an ergodic random sequence and provided that $\langle E \rangle$ has a unique minimum. Experimentally it is found that RBP, like standard back-propagation, converges robustly albeit after very many iterations. A detailed computational comparison of RBP and master/slave has yet to be performed.

The RBP algorithm is better suited for hardware implementation than the δ rule for two reasons. First, the algorithm is expressed completely in differential equations and therefore can be implemented in analog very large-scale integration. This eliminates the timing and synchronization problems which appear in digital implementations of the standard δ rule. Second, the RBP algorithm vectorizes naturally. This is because the units are homogeneous, i.e., the input, hidden, and output units all obey the same differential (difference) equations—only the components of the constant vectors I and J serve to distinguish the roles of the units.

The author wishes to acknowledge very fruitful discussions with Robert Jenkins and Ben Yuhas. Liam Healy also contributed in the early discussions. This work was supported in part by Grant No. AFOSR-87-0354 from the U.S. Air Force Office of Scientific Research.

[1]M. Minsky and S. Papert, *Perceptron* (M.I.T. Press, Cambridge, MA, 1969).

[2]Shun-Ichi Amari, in *Systems Neuroscience*, edited by Jacqueline Metzler (Academic, New York, 1977).

[3]D. E. Rumelhart, G. E. Hinton, and R. J. Williams, in *Parallel Distributed Processing*, edited by D. E. Rumelhart and J. L. McClelland (M.I.T. Press, Cambridge, MA, 1986).

[4]Shun-Ichi Amari, IEEE Trans. Systems Man Cybernet. **2**, 643–657 (1972).

[5]J. J. Hopfield, Proc. Natl. Acad. Sci. U.S.A. Bio. **81**, 3088–3092 (1984).

[6]Michael A. Cohen and Stephen Grossberg, IEEE Trans. Systems Man Cybernet. **13**, 815–826, 1983.

[7]After this Letter was submitted, the author learned of the independent work of Luis B. Almeida, who derived a discrete version of the RBP algorithm to appear in the Proceedings of the IEEE First Annual International Conference on Neural Networks, San Diego, California, June 1987, edited by M. Caudil and C. Butler (to be published).

[8]The parameters used were $\Delta t = 1$, $\alpha = 1$, $\beta = 1$, and $\eta = 0.5$. The integration was terminated when the χ^2 summed over all four patterns reached 0.1. A new pattern was presented on each iteration of Eq. (11). The difference equation corresponding to (11) was modified by the inclusion of a momentum term to accelerate the convergence.

[9]Alan Lapedes and Robert Farber, Physica (Amsterdam) **D22**, 247–259, 1986, and in *Neural Networks for Computing—1986*, edited by J. S. Denker, AIP Conference Proceedings No. 151 (American Institute of Physics, New York, 1986), pp. 283–298.

[10]The master/slave Liapunov function actually contains an extra term which adds a passive decay term to the learning equations. Our function E could be modified to include such a term, but the inclusion of such a term is not essential to the discussion.

GENERALIZATION OF BACKPROPAGATION TO RECURRENT AND HIGHER ORDER NEURAL NETWORKS

Fernando J. Pineda

Applied Physics Laboratory, Johns Hopkins University
Johns Hopkins Rd., Laurel MD 20707

Abstract

A general method for deriving backpropagation algorithms for networks with recurrent and higher order networks is introduced. The propagation of activation in these networks is determined by dissipative differential equations. The error signal is backpropagated by integrating an associated differential equation. The method is introduced by applying it to the recurrent generalization of the feedforward backpropagation network. The method is extended to the case of higher order networks and to a constrained dynamical system for training a content addressable memory. The essential feature of the adaptive algorithms is that adaptive equation has a simple outer product form.

Preliminary experiments suggest that learning can occur very rapidly in networks with recurrent connections. The continuous formalism makes the new approach more suitable for implementation in VLSI.

Introduction

One interesting class of neural networks, typified by the Hopfield neural networks [1,2] or the networks studied by Amari[3,4] are dynamical systems with three salient properties. First, they posses very many degrees of freedom, second their dynamics are nonlinear and third, their dynamics are dissipative. Systems with these properties can have complicated attractor structures and can exhibit computational abilities.

The identification of attractors with computational objects, e.g. memories and rules, is one of the foundations of the neural network paradigm. In this paradigm, programming becomes an excercise in manipulating attractors. A learning algorithm is a rule or dynamical equation which changes the locations of fixed points to encode information. One way of doing this is to minimize, by gradient descent, some function of the system parameters. This general approach is reviewed by Amari[4] and forms the basis of many learning algorithms. The formalism described here is a specific case of this general approach.

The purpose of this paper is to introduce a formalism for obtaining adaptive dynamical systems which are based on backpropagation[5,6,7]. These dynamical systems are expressed as systems of coupled first order differential equations. The formalism will be illustrated by deriving adaptive equations for a recurrent network with first order neurons, a recurrent network with higher order neurons and finally a recurrent first order associative memory.

Example 1: Recurrent backpropagation with first order units

Consider a dynamical system whose state vector x evolves according to the following set of coupled differential equations

$$dx_i/dt = -x_i + g_i(\Sigma_j w_{ij}x_j) + I_i \qquad (1)$$

where i=1,...,N. The functions g_i are assumed to be differentiable and may have different forms for various populations of neurons. In this paper we shall make no other requirements on g_i. In the neural network literature it is common to take these functions to be sigmoid shaped functions. A commonly used form is the logistic function,

$$g(\xi) = (1 + e^{-\xi})^{-1}. \qquad (2)$$

This form is biologically motivated since it attempts to account for the refractory phase of real neurons. However, it is important to stress that there is nothing in the mathematical content of this paper which requires this form -- any differentiable function will suffice in the formalism presented in this paper. For example, a choice which may be of use in signal processing is $\sin(\xi)$.

A necessary condition for the learning algorithms discussed here to exist is that the system posesses stable isolated attractors, i.e. fixed points. The attractor structure of (1) is the same as the more commonly used equation

$$du_i/dt = -u_i + \Sigma_j w_{ij}g(u_j) + K_i. \qquad (3)$$

Because (1) and (3) are related by a simple linear transformation. Therefore results concerning the stability of (3) are applicable to (1). Amari[3] studied the dynamics of equation (3) in networks with random conections. He found that collective variables corresponding to the mean activation and its second moment must exhibit either stable or bistable behaviour. More recently, Hopfield[2] has shown how to construct content addressable memories from symmetrically connected networks with this same dynamical equation. The symmetric connections in the network gaurantee global stability. The solution of equation (1) is also globally asymptotically stable if w can be transformed into a lower triangular matrix by row and column exchange operations. This is because in such a case the network is a simply a feedforward network and the output can be expressed as an explicit function of the input. No Liapunov function exists for arbitrary weights as can be demonstrated by constructing a set of weights which leads to oscillation. In practice, it is found that oscillations are not a problem and that the system converges to fixed points unless special weights are chosen. Therefore it shall be assumed, for the purposes of deriving the backpropagation equations, that the system ultimately settles down to a fixed point.

Consider a system of N neurons, or units, whose dynamics is determined by equation (1). Of all the units in the network we will arbitrarily define some subset of them (A) as *input* units and some other subset of them (Ω) as *output* units. Units which are neither members of A nor Ω are denoted *hidden* units. A unit may be simultaneously an input unit and an output unit. The external environment influences the system through the source term, I. If a unit is an input unit, the corresponding component of I is nonzero. To make this more precise it is useful to introduce a notational convention. Suppose that Φ represent some subset of units in the network then the function $\Theta_{i\Phi}$ is defined by

$$\Theta_{i\Phi} = \begin{cases} 1 & \text{if i-th unit is a member of } \Phi \\ 0 & \text{otherwise} \end{cases} \qquad (4)$$

In terms of this function, the components of the I vector are given by

$$I_i = \xi_i \Theta_{iA} \ , \qquad (5)$$

where ξ_i is determined by the external environment.

Our goal will be to find a local algorithm which adjusts the weight matrix w so that a given initial state $x^o = x(t_o)$, and a given input I result in a fixed point, $x^\infty = x(t_\infty)$, whose components have a desired set of values T_i along the output units. This will be accomplished by minimizing a function E which measures the distance between the desired fixed point and the actual fixed point i.e.,

$$E = \frac{1}{2} \sum_{i=1}^{N} J_i^2 \qquad (6)$$

where

$$J_i = (T_i - x^\infty_i)\,\Theta_{i\Omega} \ . \qquad (7)$$

E depends on the weight matrix w through the fixed point $x^\infty(w)$. A learning algorithm drives the fixed points towards the manifolds which satisfy $x_i^\infty = T_i$ on the output units. One way of accomplishing this with dynamics is to let the system evolve in the weight space along trajectories which are antiparallel to the gradient of E. In other words,

$$dw_{ij}/dt = -\eta \frac{\partial E}{\partial w_{ij}} \qquad (8)$$

where η is a numerical constant which defines the (slow) time scale on which w changes. η *must be small so that* x *is always essentially at steady state* , i.e. $x(t) \equiv x^\infty$. It is important to stress that the choice of gradient descent for the learning dynamics is by no means unique, nor is it necessarily the best choice. Other learning dynamics which employ second order time derivatives (e.g. the momentum method[5]) or which employ second order space derivatives (e.g. second order backpropagation[8]) may be more useful in particular applications. However, equation (8) does have the virtue of being the simplest dynamics which minimizes E.

On performing the differentiations in equation (8), one immediately obtains

$$dw_{rs}/dt = \eta \sum_k J_k \frac{\partial x^\infty_k}{\partial w_{rs}} \ . \qquad (9)$$

The derivative of x^∞_k with respect to w_{rs} is obtained by first noting that the fixed points of equation (1) satisfy the nonlinear algebraic equation

$$x^\infty_i = g_i(\Sigma_j w_{ij} x^\infty_j) + I_i \ , \qquad (10)$$

differentiating both sides of this equation with respect to w_{rs} and finally solving for $\partial x^\infty_k / \partial w_{rs}$. The result is

$$\frac{\partial x^\infty_k}{\partial w_{rs}} = (L^{-1})_{kr}\, g_r'(u_r) x^\infty_s \qquad (11)$$

where g_r' is the derivative of g_r and where the matrix L is given by

$$L_{ij} = \delta_{ij} - g_i'(u_i) w_{ij} \ . \qquad (12)$$

δ_{ij} is the Kroneker δ function ($\delta_{ij} = 1$ if i=j, otherwise $\delta_{ij} = 0$). On substituting (11) into (9) one obtains the remarkably simple form

$$dw_{rs}/dt = \eta\, y_r x^\infty_s \qquad (13)$$

where

$$y_r = g_r'(u_r) \sum_k J_k (L^{-1})_{kr} \ . \qquad (14)$$

Equations (13) and (14) specify a formal learning rule. Unfortunately, equation (14) requires a matrix inversion to calculate the error signals y_k. Direct matrix inversions are necessarily nonlocal calculations and therefore this learning algorithm is not suitable for implementation as a neural network. Fortunately, a local method for calculating y_r can be obtained by the introduction of an associated dynamical system. To obtain this dynamical system first rewrite equation (14) as

$$\sum_r L_{rk}\{y_r / g_r'(u_r)\} = J_k \ . \qquad (15)$$

Then multiply both sides by $f_k'(u_k)$, substitute the explicit form for L and finally sum over r. The result is

$$0 = -y_k + g_k'(u_k)\{\sum_r w_{rk} y_r + J_k\} \ . \qquad (16)$$

One now makes the observation that the solutions of this linear equation are the fixed points of the dynamical system given by

$$dy_k/dt = -y_k + g_k'(u_k)\{\sum_r w_{rk} y_r + J_k\} \ . \qquad (17)$$

This last step is not unique, equation (16) could be transformed in various ways leading to related differential equations, cf. Pineda[9]. It is not difficult to show that the first order finite difference approximation (with a time step $\Delta t = 1$) of equations (1), (13) and (17) has the same form as the conventional backpropagation algorithm.

Equations (1), (13) and (17) completely specify the dynamics for an adaptive neural network, provided that (1) and (17) converge to stable fixed points and provided that both quantities on the right hand side of equation (13) are the steady state solutions of (1) and (17).

It was pointed out by Almeida[10] that the local stability of (1) is a sufficient condition for the local stability of (17). To prove this it suffices to linearize equation (1) about a stable fixed point. The resulting linearized equation depends on the same matrix L whose transpose appears in the derivation of equation (17), cf. equation (15). But L and L^T have the same eigenvalues, hence it follows that the fixed points of (17) must also be locally stable if the fixed points of (1) are locally stable.

Learning multiple associations

It is important to stress that up to this point the entire discussionhas assumed that I and T are constant in time, thus no mechanism has been obtained for learning multiple input/output associations. Two methods for training the network to learn multiple associations are now discussed. These methods lead to qualitatively different learning behaviour.

Suppose that each input/output pair is labeled by a pattern label α, i.e. $\{I^\alpha, T^\alpha\}$. Then the energy function which is minimized in the above discussion must also depend on this label since it is an implicit function of the I^α, T^α pairs. In order to learn multiple input/output associations it is necessary to minimize all the $E[\alpha]$ simultaniously. In otherwords the function to minimize is

$$E_{total} = \sum_\alpha E[\alpha] \qquad (18)$$

where the sum is over all input/output associations. From (18) it follows that the gradient for E_{total} is simply the sum of the gradients for each association, hence the corresponding gradient descent equation has the form,

$$dw_{ij}/dt = \eta \sum_{\alpha} y^{\infty}{}_i[\alpha]\, x^{\infty}{}_j[\alpha] \quad . \tag{19}$$

In numerical simulations, each time step of (19) requires relaxing (1) and (17) for each pattern and accumulating the gradient over all the patterns. This form of the algorithm is deterministic and is guaranteed to converge because, by construction, E_{total} is a Liapunov function for equation (19). However, the system may get stuck in a local minimum. This method is similar to the master/slave approach of Lapedes and Farber[11]. Their adaptive equation, which plays the same role as equation (19), also has a gradient form, although it is not strictly descent along the gradient. For a randomly or fully connected network it can be shown that the number of operations required per weight update in the master/slave formalism is proportional to N^3 where N is the number of units. This is because there are $O(N^2)$ update equations and each equation requires O(N) operations (assuming some precomputation). On the other hand, in the backpropagation formalism each update equation requires only O(1) operations because of their trivial outer product form. Also $O(N^2)$ operations are required to precompute x^{∞} and y^{∞}. The result is that each weight update requires only $O(N^2)$ operations. It is not possible to conclude from this argument that one or the other approach will be more efficient in a particular application because there are other factors to consider such as the number of patterns and the number of time steps required for x and y to converge. A detailed comparison of the two methods is in preparation.

A second approach to learning multiple patterns is to use (13) and to change the patterns *randomly* on each time step. The system therefore receives a sequence of random impulses each of which attempts to minimize $E[\alpha]$ for a single pattern. One can then define $L(w)$ to be the mean $E[\alpha]$ (averaged over the distribution of patterns).

$$L(w) = \langle E\,[w, I^{\alpha}, T^{\alpha}\,] \rangle \tag{20}$$

Amari[4] has pointed out that if the sequence of random patterns is stationary and if $L(w)$ has a unique minimum then the theory of stochastic approximation guarantees that the solution of (13) $w(t)$ will converge to the minimum point w_{min} of $L(w)$ to within a small fluctuating term which vanishes as η tends to zero. Evidently η is analogous to the temperature parameter in simulated annealing. This second approach generally converges more slowly than the first, but it will ultimately converge (in a statistical sense) to the *global* minimum.

In principle the fixed points, to which the solutions of (1) and (17) eventually converge, depend on the initial states. Indeed, Amari's[3] results imply that equation (1) is bistable for certain choices of weights. Therefore the presentation of multiple patterns might seem problematical since in both approaches the final state of the previous pattern becomes the initial state of the new pattern. The safest approach is to reinitialize the network to the same initial state each time a new pattern is presented, e.g. $x_i(t_o) = 0.5$ for all i. In practice the system learns robustly even if the initial conditions are chosen randomly.

Example 2: Recurrent higher order networks

It is straightforward to apply the technique of the previous section to a dynamical system with higher order units. Higher order systems have been studied by Sejnowski [12] and Lee et al.[13]. Higher order networks may have definite advantages over networks with first order units alone. A detailed discussion of the backpropagation formalism applied to higher order networks is beyond the scope of this paper. Instead, the adaptive equations for a network with purely n-th order units will be presented as an example of the formalism. To this end consider a dynamical system of the form

$$dx_i/dt = -x_i + g_i(u_i) + I_i \tag{21}$$

where

$$u_i = \sum_j \cdots \sum_k w^{(n)}_{ij\cdots k} f_j(x_j) \cdots f_k(x_k) \quad . \tag{22}$$

and where there are n+1 indices and the summations are over all indices except i. The superscript on the weight tensor indicates the order of the correlation. Note that an additional nonlinear function f has been added to illustrate a further generalization. Both f and g must be differentiable and may be chosen to be sigmoids. It is not difficult, although somewhat tedious, to repeat the steps of the previous example to derive the adaptive equations for this system. The objective function in this case is the same as was used in the first example, i.e. equation (6). The n-th order gradient descent equation has the form

$$dw^{(n)}_{ij\cdots k}/dt = \eta y^{(n)\infty}{}_i f(x^{\infty}{}_j) \cdots f(x^{\infty}{}_k) \quad . \tag{23}$$

Equation (23) illustrates the major feature of backpropagation which distinguishes it from other gradient descent algorithms or similar algorithms which make use of a gradient. Namely, that the gradient of the objective function has a very trivial outer product form. $y^{(n)\infty}$ is the steady state solution of

$$dy^{(n)}_k/dt = - y^{(n)}_k + g_k'(u_k)\Big\{ f_k'(x_k) \sum_r V^{(n)}_{rk} y^{(n)}_r + J_k \Big\} \quad . \tag{24}$$

The matrix $V^{(n)}$ plays the role of w in the previous example, however $V^{(n)}$ now depends on the state of the network according to

$$V^{(n)}_{ij} = \sum_k \cdots \sum_l S^{(n)}_{ijk\cdots l} \{ f(x_k) \cdots f(x_l) \} \tag{25}$$

where is $S^{(n)}$ a tensor which is symmetric with respect to the exchange of the second index and all the indices to the right, i.e.

$$S^{(n)}_{ijk\cdots l} = w^{(n)}_{ijk\cdots l} + w^{(n)}_{ikj\cdots l} + \cdots + w^{(n)}_{ijl\cdots k} \quad . \tag{26}$$

Finally, it should be noted that: 1) If the polynomial u_i is not homogenous, the adaptive equations are more complicated and involve cross terms between the various orders and that: 2) The local stability of the n-th order backpropagation equations now depends on the eigenvalues of the matrix

$$L_{ij} = \delta_{ij} - g_i'(u_i)\, f_i'(x_i)\, V^{(n)}_{ij} \quad . \tag{27}$$

As before, if the forward propagation converges so will the backward propagation.

Example 3: Adaptive content addressable memory

In this section the adaptive equations for a content addressable memory (CAM) are derived as a final illustration of the generality of the formalism. Perhaps

the best known (and best studied) examples of dynamical systems which exhibit CAM behaviour are the systems discussed by Hopfield[1,2]. Hopfield used a nonadaptive method for programming the symmetric weight matrix. More recently Lapedes and Farber[11] have demonstrated how to contruct a master dynamical system which can be used to train the weights of a slave system which has the Hopfield form. This slave system then performs the CAM operation. The resulting weights are not symmetric.

The learning proceedure presented in this section is most closely related to the method of Lapedes and Farber in that a master network is used to adjust the weights of a slave network. In constrast to the afforementioned formalism, which requires a very large associated weight matrix for the master network, both the master and slave networks of the following approach make use of the same weight matrix. The CAM under consideration is based on equation (1). However, the interpretation of the dynamics will be somewhat different from the first section. The main difference is that the dynamics in the learning phase is constrained. The constrained dynamical system is denoted the master network. The unconstrained system is denoted the slave network. The units in the network are divided into only two sets: the set of visible units (V) and the set of internal or hidden units (H). There will be no distinction made between input and output units. Thus, I will generally be zero unless an input bias is needed in some application.

The dynamical system will be used as an autoassociative memory, thus the memory recall is performed by starting the network at a particular initial state which represents partial information about a stored memory. More precisely, suppose that there exists a subset K of the visible units whose states are known to have values T_i. Then the initial state of the network is

$$x_i(t_o) = T_i \Theta_{iK} + b_i (1-\Theta_{iK}), \tag{28}$$

where the b_i are arbitrary. The CAM relaxes to the previously stored memory whose basin of attraction contains this partial state.

Memories are stored by a master network whose topology is exactly the same as the slave network, but whose dynamics is somewhat modified. The state vector z of the master network evolves according to the equation

$$dz_i/dt = -z_i + g_i(\sum_{k=1}^{N} w_{ik} Z_k) + I_i \tag{29}$$

where Z is defined by

$$Z_i = T_i \Theta_{iV} + z_i \Theta_{iH} . \tag{30}$$

The components of Z along the visible units are just the target value specified by T. This equation is useful as a master equation because if the weights can be chosen so that the z_i of the visible units relax to the target values T_i, then a fixed point of (29) is also a fixed point of (1). It can be concluded therefore, that by training the weights of the master network one is also training the weights of the slave network. Note that the form of Z implies that equation (29) can be rewritten as

$$dz_i/dt = -z_i + g_i(\sum_{k\in H} w_{ik} z_k - \theta_i) + I_i \tag{31}$$

where

$$\theta_i = -\sum_{k\in V} w_{ik} T_k . \tag{32}$$

From equations (31) and (32) it is clear that the dynamics of the master system is driven by the thresholds which depend on the targets.

To derive the adaptive equations consider the objective function

$$E_{master} = \frac{1}{2} \sum_{i=1}^{N} J_i^2 . \tag{33}$$

where

$$J_i = Z^{\infty}_i - z^{\infty}_i . \tag{34}$$

It is straightforward to apply the steps discussed in previous sections to E_{master}. This results in adaptive equations for the weights. The mathematical details will be omitted since they are essentially the same as before, the gradient descent equation is

$$dw_{ij}/dt = \eta y^{\infty}_i Z^{\infty}_j \tag{35}$$

where y^{∞} is the steady state solution of

$$dy_k/dt = -y_k + g'_k(v_k)\{\Theta_{iH}\sum_r w_{rk} y_r + J_k\} \tag{36}$$

where

$$v_i = \sum_{i\in H} w_{ik} Z^{\infty}_k . \tag{37}$$

Equations (31), and (35)-37) define the dynamics of the master network. To train the slave network to be an autoassociative memory it is necessary to use the stored memories as the initial states of the master network, i.e.

$$z_i(t_o) = T_i \Theta_{iV} + b_i \Theta_{iH} \tag{39}$$

where b_i is an arbitrary value as before. The previous discussions concerning the stability of the three equations (1), (13) and (17) apply to equations (31) (35) and (36) as well. It is also possible to derive the adaptive equations for a higher order associative network, but this will not be done here.

Only preliminary computer simulations have been performed with this algorithm to verify their validity, but more extensive experiments are in progress. The first simulation was with a fully connected network with 10 visible units and 5 hidden units. The training set consisted of four random binary vectors with the magnitudes of the vectors adjusted so that $0.1 \le T_i \le 0.9$. The equations were approximated by first order finite difference equations with $\Delta t = 1$ and $\eta = 1$. The training was performed with the deterministic method for learning multiple associations. Figure 1. shows E_{total} as a function of the number of updates for both the master and slave networks. E_{total} for the slave exhibits discontinous behaviour because the trajectory through the weight space causes $x(t_o)$ to cut across the basins of attraction for the fixed points of equation (1).

The number of updates required for the network to learn the patterns is relatively modest and can be reduced further by increasing η. This suggests that learning can occur very rapidly in this type of network.

Discussion

The algorithms presented here by no means exhaust the class of possible adaptive algorithms which can be obtained with this formalism. Nor is the choice of gradient descent a crucial feature in this formalism. The key idea is that it is possible to express the gradient of an objective function as the outer product of vectors which can be calculated by dynamical systems. This outer product form is also responsible for the fact that the gradient can be calculated with only $O(N^2)$ operations in a fully connected or randomly connected network. In fact the number of operations per

weight update is proportional to the number of connections in the network. The methods used here will generalize to calculate higher order derivatives of the objective function as well.

The fact that the algorithms are expressed as differential equations suggests that they may be implemented in analog electronic or optical hardware.

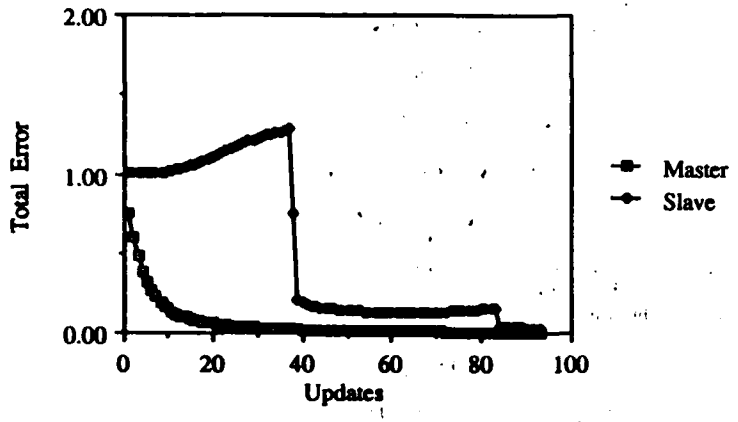

figure 1. E_{total} as a function of the the number of updates.

References

(1) J. J. Hopfield, *Neural Networks as Physical Systems with Emergent Collective Computational Abilities*, Proc. Nat. Acad. Sci. USA, Bio.79, 2554-2558, (1982)

(2) J. J. Hopfield, *Neurons with graded response have collective computational properties like those of two-state neurons*, Proc. Nat. Acad. Sci. USA, Bio. 81, 3088-3092, (1984)

(3) Shun-Ichi Amari, IEEE Trans. on Systems Man and Cybernetics, 2, 643-657, (1972)

(4) Shun-Ichi Amari, in Systems Neuroscience, ed. Jacqueline Metzler, Academic press, (1977)

(5) D. E. Rumelhart, G. E. Hinton and R.J. Williams, in Parallel Distributed Processing, edited by D. E. Rumelhart and J. L. McClelland, M.I.T. press, (1986)

(6) David B. Parker, *Learning-Logic*, Invention Report, S81-64, File 1, Office of Technology Licensing, Stanford University, October, 1982

(7) Y. LeChun, Proceedings of Cognitiva, 85, p. 599, (1985)

(8) David B. Parker, *Second Order Backpropagation: Implementing an Optimal O(n) Approximation to Newton's Method as an Artificial Neural Network*, submitted to Computer, (1987)

(9) Fernando J. Pineda, *Generalization of backpropagation to recurrent neural networks*, Phys. Rev. Lett., 18, 2229-2232, (1987)

(10) Luis B. Almeida, in the Proceedings of the IEEE First Annual International Conference on Neural Networks, San Diego, California, June 1987, edited by M. Caudil and C. Butler (to be published This is a discrete version of the algorithm presented as the first example

(11) Alan Lapedes and Robert Farber, *A self-optimizing, nonsymmetrical neural net for content addressable memory and pattern recognition*, Physica, D22, 247-259, (1986), see also, *Programming a Massively Parallel, Computation Universal System: Static Behaviour*, in Neural Networks for Computing Snowbird, UT 1986, AIP Conference Proceedings, 151, (1986), edited by John S. Denker

(12) Terrence J. Sejnowski, *Higher-order Boltzmann Machines*, Draft preprint obtained from author

(13) Y.C. Lee, Gary Doolen, H.H. Chen, G.Z. Sun, Tom Maxwell, H.Y. Lee and C. Lee Giles, *Machine Learning using a higher order correlation network*, Physica D22, 276-306, (1986)

JOURNAL OF COMPLEXITY 4, 216–245 (1988)

Dynamics and Architecture for Neural Computation*

FERNANDO J. PINEDA

Applied Physics Laboratory, Johns Hopkins University, Johns Hopkins Road, Laurel, Maryland 20707

Received April, 1987

Useful computation can be performed by systematically exploiting the phenomenology of nonlinear dynamical systems. Two dynamical phenomena are isolated into primitive architectural components which perform the operations of continuous nonlinear transformation and autoassociative recall. Backpropagation techniques for programming the architectural components are presented in a formalism appropriate for a collective nonlinear dynamical system. It is shown that conventional recurrent backpropagation is not capable of storing multiple patterns in an associative memory which starts out with an insufficient number of point attractors. It is shown that a modified algorithm can solve this problem by introducing new attractors near the to-be-stored patterns. Two primitive components are assembled into an elementary machine and trained to perform invariant pattern recognition with respect to small arbitrary transformations of the input pattern, provided the transformations are sufficiently small. The machine realizes modular learning since error signals do not propagate across the boundaries of the components.

1. INTRODUCTION

Much of the recent interest in neural computation stems from the suggestion by Hopfield (1982) that the collective properties of physical systems might be used to directly implement computational tasks. This new paradigm for computation promises to yield a new class of computing machines in which the physics of the machine and the algorithms of the computation are intimately related.

The purpose of this paper is to show how to perform useful computation by systematically exploiting the phenomenology of a class of collective dynamical systems. Initially the model-independent behavior of the dynamical systems will be discussed and it will be shown how the phenomenology of the systems can be isolated into two primitive architectural components (filters) which perform the operations of continuous nonlinear transformation and autoassociative recall. These filters are primitive in the sense that they are fundamental building blocks from which one can build hierarchical architectures.

* This work was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-87-0354 and by the Applied Physics Laboratory under IRAD-X8U.

0885-064X/88 $3.00

Backpropagation techniques for programming filters will be developed for the case of a simple model, however, the techniques apply to a broad class of neurodynamical models. The recurrent backpropagation algorithms will be presented in a formalism appropriate for implementation as a physical nonlinear dynamical system. One of the advantages of this formalism is that it uses continuous time and therefore does not exhibit certain kinds of oscillations which occur in discrete time models usually associated with backpropagation.

One of the results of the model-independent investigation will be applied to explain why the backpropagation algorithm is incapable of storing multiple patterns in a simple associative memory model. The solution to the problem will be to constrain the system during learning. The resulting algorithm not only changes the location of fixed points, it also creates new ones. This results in discontinuous learning behavior in the autoassociative memory.

As a demonstration of a simple hierarchical architecture, two primitive filters will be combined to produce an elementary pattern recognition machine. This machine is capable of recognizing patterns which have been corrupted by arbitrary transformations provided the transformations are sufficiently small. In other words the machine exhibits a limited amount of invariant pattern recognition. The two filters in the machine are capable of learning independently in the sense that error signals do not propagate across the filter boundaries. Thus the two-filter system is a simple example of the modular learning scheme proposed by Ballard (1987).

This paper is organized in the following way. In Section 2, a restricted definition of neurodynamics is given. The systems considered in this paper are subclasses of this dynamics. The way in which the dynamics is exploited to construct two kinds of filters is explained. In Section 3 the behavior of these two filters is discussed qualitatively. In Section 4 a specific neural model is chosen and discussed. This model provides a concrete system for illustrating the subsequent developments. Section 5 contains a derivation of a set of dynamical equations which are appropriate for training the model when it is used to make continuous nonlinear maps. Section 6 discusses how these equations are used to learn multiple input/output patterns. Section 7 discusses the role of time scales in the

learning dynamics. In Section 8 the dynamical equations for training an associative memory are presented and discussed. Section 9 presents a simple hierarchically organized pattern recognition system based on the components discussed in the previous sections. Finally, the results are summarized and discussed in Section 10.

2. Neurodynamics and Primitive Filters

A universally agreed-upon definition of neurodynamics does not exist, but for the purposes of analysis it is useful to define the most general features of the dynamical systems which are to be considered in this paper. The entire discussion, unless otherwise specified, will be limited to systems which have continuous-valued states and equations of motion which can be expressed as differential equations. These systems possess three general characteristics. First, they generally have very many degrees of freedom. The human brain, for example, is believed to have between 10^{11} and 10^{13} neurons (depending on which cells are counted). The state of each of these neurons can be modeled by one or more dynamical variables. It is generally believed that the computational power and fault-tolerant capabilities of neural systems results from the collective dynamics of the system. Collective effects account for the properties of many physical systems including magnetism, superconductivity, and fluid dynamics. These systems are trivial in one respect. They can all be characterized by only one or two coupling constants. Neurodynamical systems on the other hand are characterized by very many coupling constants. In general, there is a different coupling constant for each interaction. In biological systems these different coupling constants correspond to the strengths of individual synaptic junctions. A well-studied physical system which does have very many coupling constants is the spin-glass (see, e.g., Binder and Young, 1986). Not surprisingly, this system has been used as the basis for discrete neural network models, e.g., Hopfield (1982) and Hinton *et al.* (1984).

Second, the neurodynamical systems are nonlinear. Linear dynamical systems are characterized by the fact that any two solutions of the system may be added together to produce a third solution. Accordingly, linear dynamical systems can perform linear mappings only and are therefore limited in their computational ability. This is one of the implications of a rigorous analysis by Minsky and Papert (1969) of single layer networks of linear threshold units called perceptrons. Nonlinearity is a required property in associative memories if they are to distinguish between two stored patterns. This issue is discussed further in Section 9 of this paper.

The final characteristic of the neurodynamical systems is that they are

dissipative. Dissipative dynamical systems are generally described by coupled sets of first-order differential equations of the form

$$dx_i/dt = G_i(\mathbf{x}). \tag{2.1}$$

If $\mathbf{G}$ and $\mathbf{x}$ have N components then the state space of the system is N-dimensional and the trajectory of $\mathbf{x}$ is called an N-dimensional flow. A dissipative system is characterized by the convergence of the flow onto a manifold of lower dimensionality as the system evolves. General dissipative systems can exhibit complicated behavior. For example, they may converge onto one-dimensional manifolds (periodic orbits) or manifolds with fractional dimensions (strange attractors). The discussion in this paper will be confined to systems whose only behavior is to converge onto point attractors for some range of parameters and initial conditions. Point attractors are important in neural computing because the corresponding state vector values can be used to represent computational objects, e.g., memories, data structures, or rules.

Now consider a general neurodynamical system with an unspecified dependence on internal dynamical parameters, external dynamical parameters, and state variables. The system is defined by the equation

$$dx_i/dt = G_i(\mathbf{w}, \mathbf{I}, \mathbf{x}). \tag{2.2}$$

The matrix $\mathbf{w}$ represents the set of internal dynamical parameters and the vector $\mathbf{I}$ represents the set of external dynamical parameters, i.e., a control vector or external bias. It will be assumed that trajectories of this system converge onto point attractors for values of $\mathbf{w}$, $\mathbf{I}$ and initial states $\mathbf{x}^0$ in some "operating region." The concept of an operating region of the system will be taken to mean the set of $\mathbf{x}$, $\mathbf{w}$, and $\mathbf{I}$ which are permitted by the dynamics of the system, the dynamics of the learning algorithm, or the dynamics of the external environment, respectively. This concept is not sharply defined, nevertheless it is useful for describing the phenomenology of general neurodynamical systems.

Quantities evaluated at steady state will be denoted by a superscript *. In particular the point attractors will be denoted by $\mathbf{x}^*$. These are solutions of

$$0 = G_i(\mathbf{w}, \mathbf{I}, \mathbf{x}^*). \tag{2.3}$$

For a given $\mathbf{w}$ and $\mathbf{I}$, the set of initial points $\mathbf{x}^0$ that evolve to a particular fixed point is called the basin of attraction of that fixed point. The locations of the fixed points and the basin boundaries are functions of $\mathbf{w}$ and $\mathbf{I}$.

The phenomenology of the general neurodynamical system described by Eq. (2.2) can be exploited to construct filters. The term "filter" refers

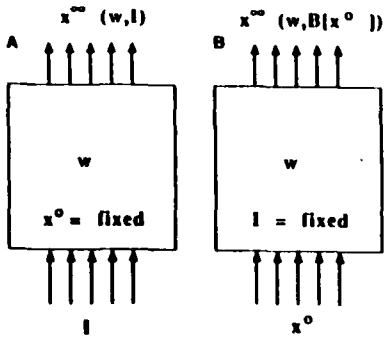

FIG. 1. For a continuous mapper (A) the filter input is the external dynamical parameter I. The output $x^\infty(w, I)$ is a continuous function of I. For an autoassociative memory (B) the filter input is the initial state x^0. The output $x^\infty(w, B[x^0])$ changes depending on which basin (denoted by B) contains the input state.

to an architectural component that performs a mapping operation from some input to some output. The architecture of a computer, the design of a program, or the organization of the brain can be described as a hierarchy of suitably designed filters. Adaptive filters, which will be discussed when learning algorithms are introduced, change their mapping operation according to the history of inputs.

If one restricts the discussion to filters with static outputs, there are two general ways of exploiting the dynamics of system (2.2) to obtain a filter. In both cases the final state x^∞ of the system is used as the output of the filter. In the first case, which is shown schematically in Fig. 1A, the bias I acts as the input to the filter. The initial state is set to some constant vector for all inputs. In the second case, which is shown schematically in Fig. 1B, the initial state x^0 of the dynamical system represents the input to the filter and it is the bias I which is set to some constant vector for all inputs.

Two well-known neural network models are examples of these two filters. The Hopfield (1984) associative memory with analog neurons is an example of the second filter. In a Hopfield network, information is stored by locating point attractors at positions in the state space which correspond to memories. The system typically converges to a complete memory if an incomplete memory, e.g., a state vector in which only some of the components are the same as a stored memory, is presented as an initial state. In general the output of such a filter is a discontinuous function of the input. For the remainder of this paper a filter which performs autoassociative recall will be called an autoassociative memory or associative memory for short. On the other hand, the feedforward network used by Rumelhart *et al.* (1986) is a limiting case of the first filter. In this case the bias I represents input into the bottom layer of the feedforward network. In general this second kind of filter will behave continuously provided that certain phenomena do not occur. These phenomena will be discussed in the following section. For the remainder of the paper this kind of filter will be called a continuous mapper or mapper for short.

The question of whether a given dynamical system can realize a given mapping is an important unanswered question. In the language of dynamics the question is whether the desired attractors are in the region of state space accessible to the system. Or, put another way, do the coupling constants exist which code the representation? This issue is beyond the scope of this paper and will not be addressed here. The approach to be followed is pragmatic: it will be assumed that a sufficiently large neurodynamical system with feedback loops is capable of representing the maps which are of interest.

Let us turn now to a discussion of how the dynamics of system (2.2) leads to either continuous or discontinuous behavior. A clear understanding of this behavior is important for the proper design of filters and learning algorithms. In the case of learning algorithms continuity is an important consideration because any learning algorithm that gradually adjusts the internal dynamical parameters is relying on the continuity of the state vector.

3. A Qualitative Description of Filter Behavior

The qualitative behavior of the continuous mapper and the autoassociative memory can be deduced from simple considerations. The presentation in this section is a schematic description of this behavior in the sense that many aspects of nonlinear dynamical systems will be simplified in the interests of clarity. For example, basins are often disconnected and have fractal boundaries. Accounting for these complications is not essential for a description of the basic phenomena. Therefore these details will be neglected. Recall that it has been assumed that the only permitted behavior of the solution of Eq. (2.2) is convergence onto point attractors, provided that the system stays within some operating region. Let us now consider the motion of these point attractors.

In a continuous mapper the location of the fixed point is usually a continuous function of w and I. The mapper will behave continuously provided that the point attractor does not vanish and provided that a basin boundary does not go past the initial point. The former situation is caused by a topological transition called a catastrophe (Arnold, 1986) which results in a discontinuous change in the attractor and basin structure (see, e.g., Amari, 1977). On the other hand, the latter situation involves no

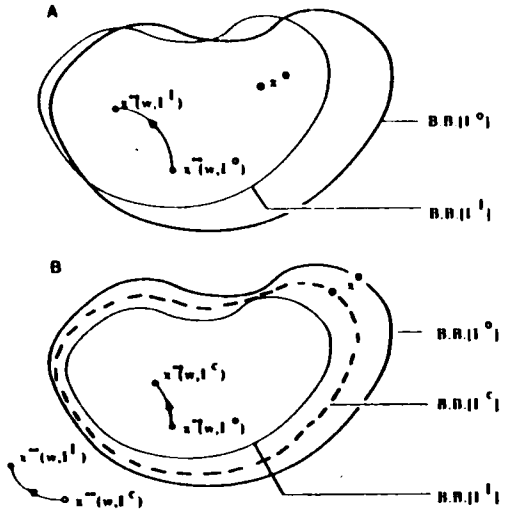

FIG. 2. If no catastrophes occur, a continuous mapper behaves continuously provided that the initial state is always inside the basin boundary (denoted by B.B.) for all values of I. This is shown in (A). On the other hand (B) shows that if the basin boundary crosses the location of the initial point, the behavior is discontinuous.

topological transition and results from the particular choice of initial state. To illustrate the latter situation consider a sequence of trials wherein one tracks the final state for a set of gradually changing inputs. The initial state is reset between trials. Figure 2A shows the schematic behavior of the final state and the basin boundary. Let the interior of the boundary represent one basin and the exterior represent another basin. If w is fixed and I changes continuously from $\mathbf{I}^0$ to $\mathbf{I}^1$, the fixed point moves continuously from $\mathbf{x}^*(w, \mathbf{I}^0)$ to $\mathbf{x}^*(w, \mathbf{I}^1)$. The boundary moves also, but at no time does the boundary cross the initial point $\mathbf{x}^0$. Thus $\mathbf{x}^*$ depends continuously on I. On the other hand consider Fig. 2B. In this case the fixed point moves continuously until the I reaches the value $\mathbf{I}^c$. As I goes through the value $\mathbf{I}^c$ the basin boundary goes past the initial state $\mathbf{x}^0$ and the initial point is suddenly in the exterior basin. Accordingly the steady state solution jumps discontinuously to the point attractor of this basin. One can turn the argument around and consider changing $\mathbf{x}^0$ while keeping I fixed. This is the case to consider if one is presenting multiple inputs. Suppose one examines $\mathbf{x}^*$ only whenever a specific input, say $\mathbf{I}^\alpha$, is presented to the system. Furthermore, suppose the "initial" state is not reset for each new pattern but instead it is taken to be the steady state from the previously presented pattern. Then, if a catastrophe does not occur, the final state for

pattern $\mathbf{I}^\alpha$ will be unique in the operating region, provided that none of the final states for the other patterns is outside the basin boundary for $\mathbf{I}^\alpha$.

Similar arguments apply when one considers the qualitative behavior of the continuous mapper in the learning process. The only difference is that w is varied gradually. To be more precise it changes slowly compared to the relaxation time of x. Therefore the instantaneous x can be approximated by the steady state solution of Eq. (2.2). In physics this is known as the adiabatic approximation. Suppose one is training the system on a single pattern by changing w adiabatically. Then I is a constant for all time the system is always in steady state. By definition, the fixed point is always inside its basin so the only way a discontinuity can occur is for a catastrophe to occur. Barring such an occurrence the steady state solution will move gradually and continuously toward the desired final state. Now suppose one is training the system on multiple patterns so that I is no longer a constant for all time but instead makes transitions between some set of input vectors. If the transitions occur adiabatically then only discontinuities due to catastrophes can occur. If, on the other hand, the transitions occur suddenly, as is the case in most neural network simulations, then discontinuities due to basin boundaries crossing "initial" states can occur, where the "initial" state is the steady state solution for the previously presented input. The general conclusion is that for the mapper to have continuous outputs and for learning to take place, the accessible states of the system should be in a region of the state space which has no basin boundaries for all w and I which will be encountered. This is the case if the point attractor is unique for fixed w and I.

Associative memories have completely different requirements. I is a fixed constant and for the purposes of this discussion it can be taken to be zero without loss of generality. Accordingly the point attractors and basin boundaries depend on w alone. An associative memory relies on having very many basins distributed over the operating region of the state space. Each of the basins is associated with a memory. If the untrained associative memory does not have enough attractors it may be difficult to store memories. To see this consider Fig. 3. Suppose one has a learning algorithm which moves attractors by adiabatically changing the internal parameters w in response to some measure of the separation between the location of the desired memory and the point attractor. Then, if there is more than one to-be-stored memory within a basin of the untrained system, the learning algorithm may not be able to store both memories because they both converge to the same final state. To store the memories the learning algorithm must somehow get the two memories into different basins by either going through a catastrophe or by a basin boundary going across a to-be-stored memory. The latter cannot be guaranteed to occur in a learning algorithm which responds to the location of a point attractor

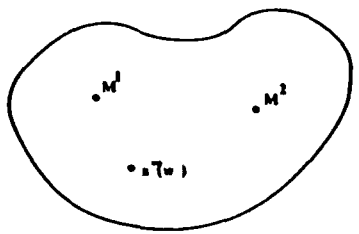

FIG. 3. Two memories, M^1 and M^2 are in the basin of a single point attractor $X^*(w)$. This point attractor cannot be moved to two locations by adjusting w. Either a catastrophe must occur so that the memories end up in the basins of different point attractors or else the basin boundary must go across one of the memories and thereby place the memory in a different basin.

only, because such a learning algorithm does not have direct control over the basin boundaries.

The general conclusion of this analysis is that continuous mappers and associative memories have conflicting requirements. For fixed I and w, the continuous mapper requires a unique point attractor in the operating region whereas the associative memory requires multiple point attractors. This conflict must be resolved if one is to build and train hierarchical systems with both filters. A specific example of this conflict and a way of circumventing it is given in Section 8.

4. A Simple Nonlinear Neural Model

The discussion now narrows to a specific neural model. The intent is to use a simple model so as not to obscure the subsequent discussions with mathematical complications. The reader should keep in mind that the techniques discussed throughout this paper apply equally well to high-order models (Pineda, 1987b).

A simple nonlinear model for an interacting system of N neurons which ignores propagation time delays between neurons is specified by the system of equations

$$du_i/dt = -u_i + \sum_j w_{ij} g(u_j) + I_i, \tag{4.1}$$

where u is the state vector of the system. The summation convention in this and all other equations in the paper is that the sum runs from $j = 1$ to N where N is the number of nodes. The weight-matrix elements w_{ij} represent the synaptic strengths of the connections between the neurons and I_i represent external biases. A commonly used form for the function g is the logistic function,

$$g(u) = (1 + e^{-u})^{-1}. \tag{4.2}$$

This simple model is well studied and is the basis of many current neural network models. The biological motivation for this model is reviewed briefly by Sejnowski (1981) and an interpretation in terms of electronic components is given by Hopfield (1984).

This author is aware of three useful operating regions for the system given in Eq. (4.1). First, when the weight matrix is lower triangular the network is of the feedforward type. Hence, there are no recurrent loops and it converges to a unique point attractor for all initial conditions. Second, when the weight matrix is symmetric, the system possess a Liapunov function and it must converge to one of many point attractors (Hopfield, 1984) and (Cohen, 1983). Finally, Atiya (1987) has shown that the system always converges to a unique point attractor provided that

$$\sum_i \sum_j w_{ij}^2 < 1/(\max_i |g_i'|)^2. \tag{4.3}$$

It is doubtful that these three cases exhaust the possible operating regions. Indeed, arguments given by Geman (1981) suggest that the system will converge "almost always" to a point attractor provided that a chaotic hypothesis holds. The chaotic hypothesis allows the weights and the activations to be treated as independent random variables (Amari, 1972).

5. Learning in a Continuous Mapper

Now let us turn to a specific technique for programming the dynamics of system (4.1). The learning algorithms to be presented are continuous and recurrent generalizations of the backpropagation algorithms discussed by Parker (1982), Le Cun (1985), and Rumelhart *et al.* (1986). Backpropagation is a very useful and general tool for training neural network models with arbitrary connectivity. The literature is now rich with recurrent, higher order, and stochastic variations of the original algorithms, e.g., Almeida (1987), Atiya (1987), Parker (1987), Rohwer and Forrest (1987) and Samad and Harper (1987), to name a few. It is worthwhile to point out that the first-order finite difference approximation of

Eq. (4.1), with $\Delta t = 1$, is the form of the model which is often associated with backpropagation. In fact, the difference equations suffer from oscillations which are artifacts of the discrete time approximation. This is discussed briefly in Appendix C.

An algorithm for training system (4.1) which allowed recurrent connections was first introduced by Lapedes and Farber (1986a, 1986b). This algorithm did not use the backpropagation technique for calculating the gradient. A more efficient algorithm which used backpropagation to reduce the amount of calculation was introduced by Pineda (1987a). A later paper by Pineda (1987b) emphasized that the method for obtaining the algorithm was quite general and applied to an entire class of neural network models, including higher order networks. This formalism is now reviewed.

The adaptive dynamics adjusts the weights w of the dynamical system (4.1) when it is used as a continuous mapper. Rather than using the form (4.1) it has been found convenient to transform these equations into the form

$$dx_i/dt = -x_i + g_i\left(\sum_j w_{ij}x_j + I_i\right). \tag{5.1}$$

Equation (4.1) is transformed into (5.1) by an affine transformation of the form $\mathbf{u} = \mathbf{w}\mathbf{x} + \mathbf{I}$. If w is nonsingular the transformation is invertible and the attractor structure of Eq. (4.1) is the same as Eq. (5.1). If g_i is the logistic function, x_i is bounded between $0 \le x_i \le 1$ and the operation region is containing within the unit N-cube.

To simplify the subsequent mathematical manipulations it is useful to introduce a notational convention. Suppose that Φ represents some subset of units in the network. Then the set function $\Theta_{i\Phi}$ is defined by

$$\Theta_{i\Phi} = \begin{cases} 1 & \text{if } i\text{th unit is a member of } \Phi \\ 0 & \text{otherwise.} \end{cases} \tag{5.2}$$

In the continuous mapper there are three important subsets of units. The first is the subset (A) of input units. The second is the subset (Ω) of output units. Note that a unit can be simultaneously an input unit and an output unit. Finally, the set (H) of hidden units contain all the units which are neither members of A nor Ω.

The external environment influences the system through the bias term, I. If the ith unit is an input unit then $I_i = \xi_i$, otherwise $I_i = 0$, where ξ_i is the value of the external input. This is expressed concisely by the following relation

$$I_i = \xi_i\Theta_{iA}. \tag{5.3}$$

Initially it will be assumed that the inputs ξ_i and outputs η_i are constants in time, thus only a single input/output association (or pattern) is considered. The goal will be to find a local algorithm which adjusts the weight matrix w so that a given initial state $\mathbf{x}^0$ and a given set of inputs ξ_i result in a point attractor, the components of which have a desired set of values η_i on the output units, i.e.,

$$\eta_i = x_i^*\Theta_{i\Omega} \tag{5.4}$$

along the output units. This will be accomplished by minimizing a function E which measures the Euclidean distance between the desired position of the point attractor and the actual position of the point attractor. This function is

$$E = \frac{1}{2}\sum_i J_i^2, \tag{5.5}$$

where

$$J_i = (\eta_i - x_i^*)\Theta_{i\Omega}. \tag{5.6}$$

The function E depends on the weight matrix w through the fixed point $\mathbf{x}^*(\mathbf{w})$. A dynamical way of minimizing E is to let the system evolve in the weight space along a "learning trajectory" which descends against the gradient of E. Thus the equation of motion for a weight matrix element w_{rs} is

$$\tau_w dw_{rs}/dt = -\frac{\partial E}{\partial w_{rs}}, \tag{5.7}$$

where τ_w is a numerical constant which defines the (slow) time scale over which w changes. τ_w must be large so that the weights change adiabatically. If this condition is not satisfied then E is not a function of the point attractor.

It is important to stress that the system evolves both in the space of activations (state space) and in the space of weights (weight space or parameter space). The evolution in the state space is determined by Eq. (5.1) whereas the evolution in the parameter space is determined by Eq. (5.7).

The choice of Eq. (5.7) for the learning dynamics is by no means unique, e.g., Lapedes and Farber (1986b). Other learning dynamics which

employ second-order time derivatives, e.g., the momentum method (Rumelhart *et al.*, 1986) or which employ second-order space derivatives (Parker, 1987) may be more useful in particular applications or hardware implementations. Equation (5.7) does have the virtue of having the simplest functional form which minimizes E by use of a gradient.

The remainder of this section discusses the derivation of a dynamical neural algorithm for efficiently calculating the gradient. Begin by performing the differentiations in Eq. (5.7). One immediately obtains

$$\tau_w dw_{rs}/dt = \sum_k J_k \frac{\partial x_k^\infty}{\partial w_{rs}}. \tag{5.8}$$

The derivative of x_k^∞ with respect to w_{rs} is obtained by first noting that $\mathbf{x}^\infty$ is a fixed point of Eq. (5.1) and hence the kth component must satisfy the nonlinear algebraic equation

$$x_i^\infty = g_i\left(\sum_j w_{ij} x_j^\infty + I_i\right). \tag{5.9}$$

Upon differentiating both sides of this equation with respect to w_{rs} and solving for $\partial x_k^\infty/\partial w_{rs}$ one obtains

$$\frac{\partial x_k^\infty}{\partial w_{rs}} = (\mathbf{L}^{-1})_{kr} g_r'(u_r^\infty) x_s^\infty, \tag{5.10}$$

where

$$u_r = \sum_j w_{rj} x_j + I_r. \tag{5.11}$$

The details of the derivation of Eq. (5.10) are given in Appendix A. The function g_i' is the derivative of g_i and the elements of the matrix $\mathbf{L}$ are given by

$$L_{ij} = \delta_{ij} - g_i'(u_i^\infty) w_{ij}, \tag{5.12}$$

where δ_{ij} are the elements of the identity matrix. On substituting (5.10) into (5.8) one obtains the simple outer product form

$$\tau_w dw_{rs}/dt = y_r^\infty x_s^\infty, \tag{5.13}$$

where y_r^∞ is defined by

$$y_r^\infty = g_r'(u_r^\infty) \sum_k J_k (\mathbf{L}^{-1})_{kr}. \tag{5.14}$$

Equations (5.13) and (5.14) specify a formal learning rule for modifying the weights. Unfortunately, Eq. (5.14) requires a matrix inversion to calculate the "error signals" y_r^∞. Direct matrix inversions are necessarily nonlocal calculations and therefore this learning algorithm is not suitable for implementation as a neural network. A local method for calculating y_r^∞ can be obtained by the introduction of an associated dynamical system. To obtain this dynamical system first rewrite Eq. (5.14) as

$$\sum_r \mathbf{L}_{rk}\{y_r^\infty / g_r'(u_r^\infty)\} = J_k. \tag{5.15}$$

Then multiply both sides by $g_k'(u_k^\infty)$ and substitute the explicit form for $\mathbf{L}$. Finally, sum over r. The result is

$$0 = -y_k^\infty + g_k'(u_k^\infty)\left\{\sum_r w_{rk} y_r^\infty + J_k\right\}. \tag{5.16}$$

One now makes the observation that the solutions of this linear equation are the fixed points of the dynamical system given by

$$dy_k/dt = -y_k + g_k'(u_k^\infty)\left\{\sum_r w_{rk} y_r + J_k\right\}. \tag{5.17}$$

The reader familiar with numerical techniques will recognize that this method of obtaining y_k^∞ is little more than a relaxation method for inverting a matrix. It is convenient to borrow the terminology of feedforward networks and refer to Eq. (5.1) as the forward propagation equation and to Eq. (5.17) as the backward propagation equation.

Equations (5.1), (5.13), and (5.17) completely specify the dynamics for an adaptive neural network, provided that (5.17) converges to point attractors. Almeida (1987) has pointed out that the local stability of (5.1) is a sufficient condition for the local stability of (5.17). The proof of this result is given in Appendix B.

The objective function which was chosen is based on Euclidean distance. The most general objective function, however, only has to be separable and bounded from below if it is to lead to local equations. If the function is separable it has the form

$$I = \sum_i e_i(x_i). \tag{5.18}$$

The only change to the adaptive equations is in the definition of J_i which generally has the form

$$J_i = -\partial e_i/\partial x_i. \tag{5.19}$$

A useful objective function based on probability is discussed by Baum, (1987).

Recall that up to this point the entire discussion has assumed that ξ and η are constants in time. Thus, no mechanism has been obtained for learning multiple input/output associations. The question of how to learn multiple patterns is addressed in the next section.

6. Learning Multiple Patterns

A method for learning multiple patterns in a gradient descent algorithm was suggested by Amari (1977). His solution was to present the patterns randomly to the network. This corresponds to solving (5.1), (5.13), and (5.17) simultaneously as the patterns change randomly with time. Therefore, the system is subject to a sequence of random forces, each of which attempts to minimize the error for a single pattern. Under certain technical conditions this may result in convergence onto a global minimum. The precise statement is the following. Suppose that each input/output pair is labeled by a pattern label α, i.e., $\{\eta^\alpha, \xi^\alpha\}$. Then define the quantity $E_m(\mathbf{w})$ to be the error $E[\alpha]$ averaged over the distribution of patterns, i.e.,

$$E_m(\mathbf{w}) = \langle E[\mathbf{w}, \xi^\alpha, \eta^\alpha] \rangle. \tag{6.1}$$

If the sequence of random patterns is stationary and if the function $E_m(\mathbf{w})$ has a unique minimum then the theory of stochastic approximation guarantees that the solution of a gradient descent equation will converge to the minimum point $\mathbf{w}_{min}$ of $E_m(\mathbf{w})$ to within a small fluctuating term which vanishes as $1/\tau_w$ tends to zero.

Random presentation has a built in mechanism for climbing out of local minima of $E_m(\mathbf{w})$ which suggests that it might be able to converge to the global minima of $E_m(\mathbf{w})$. White (1987) has concluded in a recent analysis that the existence of nonunique global minima in neural networks violates one of the assumptions of the convergence proof and therefore the method of random presentation is essentially a method for finding local minima only. It will, however, find the deepest minimum within some (unspecified) neighborhood.

The random presentation of patterns requires that each pattern be presented for a given amount of time. The length of this time influences

whether the patterns are learned or not. This is a special case of the more general issue of the role of time scales in the adaptive network. Let us now turn to a discussion of time scales.

7. Time Scales and Learning

For the system to learn it is necessary for the time scales to be properly chosen. To examine this it is useful to write the dynamical equations with explicit time scales. The equations are

$$\tau_x dx_i/dt = -x_i + g_i(u_i) + I_i(t/\tau_P), \tag{7.1}$$

$$\tau_y dy_k/dt = -y_k + g_k'(u_k)\Big\{\sum_r w_{rk} y_r + J_k(t/\tau_P)\Big\} \tag{7.2}$$

and

$$\tau_w dw_{ij}/dt = y_i x_j. \tag{7.3}$$

The relaxation time scale of the forward propagation is τ_x. The relaxation time scale of the backward propagation is τ_y and the adaptation time scale of the system is τ_w. It is straight forward to establish relationships which must be satisfied by the characteristic time scales of the system. First, note that $y_i x_j$ is the correct form of the gradient only if x and y are the steady state solutions of Eqs. (7.1) and (7.2). This will hold if the system parameters w and I change adiabatically. This condition constrains the time scales.

Let τ_P be the characteristic time scale over which the input and output patterns fluctuate. Then, for Eqs. (7.1) and (7.2) to operate near steady state it is necessary that the solution relax with a characteristic time much faster than τ_P, i.e., $\tau_P \gg \tau_x$ and $\tau_P \gg \tau_y$. Also, for Eqs. (7.1) and (7.2) to operate near steady state it is necessary for w to change slowly relative to the relaxation times of x and y; thus one must also have $\tau_w \gg \tau_x$ and $\tau_w \gg \tau_y$. Next, for the flow of Eq. (7.2) to be locally stable it is necessary for the right hand side of Eq. (7.2) to correspond to the transpose of the linearized Eq. (7.1) (see Appendix B). This is guaranteed if $\tau_y \gg \tau_x$.

The final constraint follows from the requirement that the system be able to learn multiple patterns. If the relaxation time of the weights is less than the characteristic time of the pattern fluctuations then the system will trivially learn and then forget each subsequent pattern. To keep the weights from tracking the fluctuations it is necessary that $\tau_w \gg M\tau_P$, where M is the number of patterns. These relationships imply that

$$(\tau_w/M) \gg \tau_P \gg \tau_y \gg \tau_x. \tag{7.4}$$

In practice it is found that these relationships need not be strictly satisfied for the system to learn.

8. Learning in an Autoassociative Memory

Now let us consider how to program autoassociative memory. Recall that a filter which performs autoassociative recall uses the initial state of the system as the input and the final state as the output. Therefore the set of input units and the set of output units are one and the same. This set of units will be called the set of visible units (V). All the other units are hidden (internal) units and the corresponding set is denoted by (H). With this notation the initial state of the system is simply

$$x_i^0 = \eta_i \Theta_{iV} + b_i \Theta_{iH}, \tag{8.1}$$

where the b_i are arbitrary constants and the η_i are components of a memory cue.

Recall also that the input bias $\mathbf{I}$ is an arbitrary constant vector for all patterns. Accordingly it is set equal to zero without loss of generality. Thus the dynamical equations have the form

$$dx_i/dt = -x_i + g_i\left(\sum_k w_{ik} x_k\right). \tag{8.2}$$

The goal of the learning algorithm is to position the point attractors of this dynamical system at the locations of the to-be-stored memories. Thus the point attractors must satisfy

$$\eta_i^\alpha = x_i^{*\alpha}(\mathbf{w}) \Theta_{iV} \tag{8.3}$$

for all patterns α. One might think it possible to train the associative memory with essentially the same gradient descent algorithm as in Section 5 except with the modification of resetting the state $\mathbf{x}$ for each memory instead of resetting $\mathbf{I}$. This investigator has never succeeded in training a network this way. The reason for this breakdown is the mechanism described in Section 3. All the initial states are in a single basin and all the trajectories converge onto a single degenerate point attractor. This attractor is a function of $\mathbf{w}$ only. The system output becomes the mean of all the memories since the function being minimized is

$$E(\mathbf{w}) = \frac{1}{2} \sum_\alpha \sum_{i \in V} [\eta_i^\alpha - x_i^*(\mathbf{w})]^2, \tag{8.4}$$

where α is a pattern label. The minimum of this function occurs when the point attractor $\mathbf{x}^*(\mathbf{x})$ equals the average output pattern, e.g., when $\langle \eta_i^\alpha \rangle = x_i^\alpha$. Clearly this is a local minimum of the objective function. The random presentation method discussed in Section 6 might enable the system to climb out of the minimum after a sufficient number of presentations but this cannot be guaranteed. A deterministic algorithm has no mechanism for escape.

This problem can be circumvented by constraining the network during learning. As shall now be shown this leads to a dynamical system with different point attractors for each to-be-stored memory. To make sure that there is no possibility of confusion, the dynamical variables in the constrained system are called $\mathbf{z}$ rather than $\mathbf{x}$. Accordingly consider the constrained system

$$dz_i/dt = -z_i + g_i\left(\sum_k w_{ik} Z_k\right), \tag{8.5}$$

where $\mathbf{Z}$ is defined to be

$$Z_i = \eta_i \Theta_{iV} + z_i \Theta_{iH}. \tag{8.6}$$

As in Section 5 the discussion will first consider the storage of a single memory which is represented by η_i.

To see that constraining the system has broken the degeneracy of the point attractors substitute Eq. (8.6) into (8.5) to obtain

$$dz_i/dt = -z_i + g_i\left(\sum_{k \in H} w_{ik} z_k + I_i\right), \tag{8.7}$$

where

$$I_i = \sum_{k \in V} w_{ik} \eta_k. \tag{8.8}$$

The point attractors of Eq. (8.7) (if they exist) are again functions of an externally determined bias vector $\mathbf{I}$. This breaks the degeneracy of the attractors and it is again possible to train the system on multiple patterns. The effect is to transform an associative memory into a continuous mapper during the training process. As before it will be assumed that the constrained system has only point attractors in its operating region.

Equation (8.7) is useful for training (8.2), because if the weights can be adapted so that the visible units relax to the correct memories, then a fixed point of (8.7) is also a fixed point of (8.2). Therefore, by training the

constrained system one is simultaneously training the unconstrained system. The local stability of a fixed point in the constrained system does not imply the local stability of the same fixed point in the unconstrained system. Nevertheless, in practice the algorithm seems to produce stable fixed points. Recently it has been shown that memories are stored when unstable fixed points become stable (Simard, 1988). This is an issue which needs to be investigated more carefully.

The steps required to derive the new adaptive equations are similar to the steps in Section 5 except that one considers the new objective function

$$E_c = \frac{1}{2}\sum_i J_i^2, \tag{8.9}$$

where

$$J_i = Z_i^x - z_i^x. \tag{8.10}$$

The mathematical details will be omitted since they are essentially the same as in Section 5. The new gradient descent equation is

$$\tau_w dw_{ij}/dt = y_i^x Z_j^x, \tag{8.11}$$

where y^x is the steady state solution of

$$dy_k/dt = -y_k + g_k'(v_k^x)\left\{\Theta_{kH}\sum_r w_{rk}y_r + J_k\right\} \tag{8.12}$$

and where

$$v_i^x = \sum_{k\in H} w_{ik}z_k^x + I_i. \tag{8.13}$$

Equations (8.7), (8.11), and (8.12) define the dynamics of the constrained network. The previous discussions concerning the stability of the three Eqs. (5.1), (5.14), and (5.17) apply to Eqs. (8.7), (8.11), and (8.12) as well. The discussions in Sections 6 and 7 concerning multiple patterns and time scales apply to these equations as well.

Once the weights are determined, memory recall is performed by starting the unconstrained network in a state which represents a partial memory cue. The system converges to the previously stored memory whose basin of attraction contains this initial state. It is also conceivable that the system could converge to a spurious memory, i.e., a point attractor that does not correspond to a stored memory. In practice this occurrence has not been observed by this investigator.

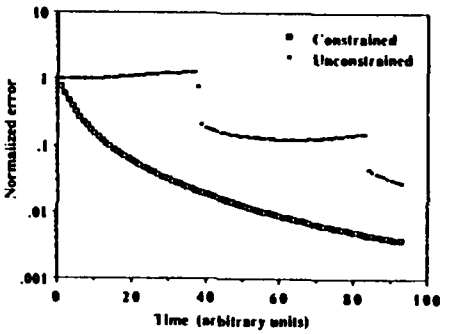

FIG. 4. Typical discontinuous behavior in the associative memory when it is unconstrained and tested at each time step in a learning process. The network contained 10 visible units, 5 hidden units, and 225 connections. The training set consisted of four arbitrarily selected binary vectors with the magnitudes of the vectors adjusted so that $0.1 \le T_i \le 0.9$. Learning was performed using the deterministic method in Appendix C which also contains the definition of normalized error.

The learning behavior of this associative memory is remarkable and novel. If the associative memory is tested at each time step during the training process by unconstraining it and presenting all the patterns, it is seen that at some time the unconstrained system spontaneously jumps to a point attractor near one of the to-be-stored memories. This occurs for each of the remaining to-be-stored memories in turn, provided that there are not too many of them. Figure 4 illustrates this behavior in a simple case in which four memories are stored. The error function of the unconstrained system undergoes discontinuous jumps as the degeneracy of the single final state is continually broken until there is one final state for each of the four to-be-stored memories. The mechanism for breaking the degeneracy can be (1) a catastrophe in the neighborhood of the to-be-stored memory, i.e., the spontaneous creation of a point attractor, (2) a basin boundary going past a to-be-stored memory, i.e., an already existing attractor is brought into the operating region, or (3) both of the above. The precise mechanism is currently a subject of investigation.

9. A Simple Hierarchical System

This section is concerned with an elementary hierarchical system. The primitive components in this hierarchy are the filters whose behavior and programming has been the focus of the above discussions.

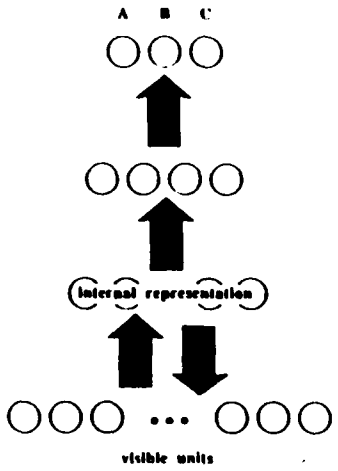

FIG. 5. This shows the topology of the two-filter machine. The layers have 1320 units, 50 units, 4 units, and 3 units, respectively. The first two layers are connected in both directions randomly with approximately 20% sparsity, so there are 25,406 total connections. The remaining layers are fully connected. There are no connections within a layer. The state of each node is determined by an equation of the form (5.1).

Large neural networks based on backpropagation do not scale well, i.e., the convergence time for learning grows faster than the number of layers. Ballard (1987) has suggested that this problem can be overcome by isolating the learning to individual modules. The filters described in this paper can be used to realize this hierarchical approach. The associative memory presented here forms internal representations when provided with hidden units. As suggested by Ballard, these internal representations can be passed to other modules in the hierarchy. In particular suppose the internal representation from an associative memory is passed to a continuous mapper. This mapper can be trained to map this representation to some external representation. The resulting two-filter machine is a primitive pattern recognition device. Figure 5 shows the topology for a four-layer implementation of a two-filter machine. The first layer is the visible layer of an autoassociative memory. The second layer is simultaneously the internal layer of the memory and the input layer of a three layer feedforward mapper. As required by Ballard, error signals are not propagated across the boundary of the two filters. Therefore the associative memory can learn independently of the continuous mapper. Furthermore the continuous mapper can be trained simultaneously with the associative memory or it can be trained after the associative memory has been trained. This network has been trained to recognize the three digitized

FIG. 6. Three images learned by the pattern recognition machine. Each image consists of 55 × 24 eight-bit pixels. Each image was preprocessed so as to have a mean pixel intensity of 0.5 and a standard deviation of 0.2.

images shown in Fig. 6. (More images can be stored by increasing the connectivity.) Only one of the three output units, denoted A, B, and C, turns on, depending on which image is present.

Despite its simplicity this system is capable of a limited amount of invariant visual pattern recognition, i.e., the output of the trained machine is invariant with respect to small arbitrary transformations of the input image. The invariance is due to the fact that any transformation of a stored pattern can be described as a displacement away from the corresponding fixed point. If this displacement is sufficiently small, the displaced point will not change basins and the associative memory filter will converge to the stored pattern. The amount of allowed displacement is difficult to quantify since it depends on the detailed shape of the basin boundaries. Nevertheless these displacements can be large in a perceptual sense as is seen in Figs. 7A–7C. These examples are correctly recognized by the network and illustrate a limited degree of invariant pattern recognition with respect to obscuration, translation, and noise. From the previous discussion it is also clear that the system will exhibit invariant pattern recognition with respect to small rotations and small scale changes. Furthermore the system will also disambiguate two patterns presented simultaneously. This latter feature is a direct consequence of the nonlinear nature of the associative memory and deserves a brief discussion.

FIG. 7. Three images used to demonstrate invariant pattern recognition. Image (A) is partially obscured. Image (B) has been shifted to the left by approximately 10% (2 pixels). Image (C) has uniform noise in the range ± 0.5 added to it.

The role played by nonlinearity in the associative memory filter becomes clear when one compares the behavior of a dynamical associative memory with a linear associative memory, e.g., Kohonen's model (1984). Consider the case where the memory cue consists of a linear combination of two stored memories. In this case the nonlinear dynamical memory will converge onto the dominant memory provided the contribution from the secondary memory is sufficiently small. More precisely any linear combination of two memories can be represented as a point on a line segment joining the two memories in the state space. If the basins are not overly convoluted the dominant memory will be recalled reliably. If the basins are overly convoluted the dominant memory will be recalled only if the contribution from the secondary memory is sufficiently small. On the other hand, the linear associative memory cannot separate the two images because the recall is based on a projection operator. This linear operator simply projects the initial state (the memory cue) onto the subspace S of the state spanned by the stored memories. Now suppose that the initial state is the sum of a stored memory $\mathbf{x}$, plus a perturbation ε, i.e., $\mathbf{x} = \mathbf{x}_i + \varepsilon$. The projection operator will be able to retrieve the "correct" memory, only to the extent that ε is contained in the space orthogonal to S. On the other hand, if ε is contained in S the projection operator simply performs the identity operation. One concludes that a linear associative memory is not an appropriate input filter for a pattern recognition neural network.

The recognition task performed by this simple system could be performed by an associative memory alone by adding three visible units which label each picture. However, this would not have served the purpose of demonstrating hierarchical architecture. Furthermore, since the simple system described here already exhibits, to a limited degree, many of the properties which are important for practical pattern recognition devices, it is not unreasonable to assume that by building up a suitable hierarchy of filters, a system with more robust invariant pattern recognition properties will emerge (e.g., Fukushima (1987)).

10. Discussion and Conclusions

This paper has presented a systematic approach for exploiting the dynamics of a general class of neurodynamical systems for the purpose of neural computation. The starting point was an understanding of the model-independent properties of these systems. Two filters were identified, one which performed continuous nonlinear transformations and another which performed autoassociative recall. It was shown that the continuous mapper and the autoassociative memory make conflicting demands on the neurodynamical system. The former requires a unique attractor in the operating region whereas the latter requires multiple attractors in the operating region.

It was shown that these conflicting demands cause the failure of conventional recurrent backpropagation when an attempt is made to store multiple patterns in an associative memory which starts with an insufficient number of point attractors. A backpropagation technique was developed for the associative memory which effectively converts it into a continuous mapper during the training process. This results in an algorithm which introduces additional fixed points near the to-be-stored memories.

The identification of primitive filters and the development of programming techniques for them is a first step in a systematic approach for the construction of hierarchically organized networks. This approach was demonstrated by the construction of a simple hierarchically organized network for pattern recognition. The simple system exhibited a limited degree of invariant pattern recognition.

There are several outstanding research questions which still need to be addressed: first is the question of stability. It is not difficult to start a recurrent network with weights which satisfy one or another of the stability constraints which are given in Section 4. However, it is usual for the weights to eventually violate these constraints as the system learns. Nevertheless, this investigator rarely experiences the onset of oscillations in any of his simulations. It is difficult to believe that this is due to chance.

Since the onset of instability must be associated with a catastrophe it seems reasonable to conjecture that a detailed study of Eq. (5.1) might reveal conditions under which catastrophes cannot occur. This could explain the remarkable stability of the trained networks. Work on this question is underway.

A second research question is how to interface the various filters. This was not a problem in the example given in the previous section. A more difficult case occurs when one wishes to connect two associative memories. The question is: "How does one reset the memory in the second layer of a hierarchy?" One solution is to introduce time-dependent oscillations to latch the output of one filter into the input of another. Such a mechanism would introduce a system-wide "clock." This solution is theoretically feasible, but the question is open as to whether this is a desirable solution for physical dynamical systems.

In closing, it is the belief of this investigator that the promise of neural computation will be realized when two lines of research converge. One line of research, which is the subject of this paper, is to understand the underlying dynamics and architecture to the point that it becomes possible to model neural computational systems in a simple and systematic way. The second line of research is to understand how to exploit the properties of real collective physical systems to implement these models in native hardware.

Appendix A

In this appendix the steps required to derive Eq. (5.10) are given. Begin by differentiating Eq. (5.19) with respect to w_{rs} on both sides. The result is

$$\frac{\partial x_i^*}{\partial w_{rs}} = g_i'(u_i^*) \sum_j \left\{ \frac{\partial w_{ij}}{\partial w_{rs}} x_j^* + w_{ij} \frac{\partial x_j^*}{\partial w_{rs}} \right\}. \tag{A.1}$$

Now, since the elements of the matrix w are independent it follows that the partial derivative $\partial w_{ij}/\partial w_{rs}$ is one if and only if $i = r$ and $j = s$ and zero otherwise, i.e.,

$$\frac{\partial w_{ij}}{\partial w_{rs}} = \delta_{ir}\delta_{js}, \tag{A.2}$$

where δ_{ij} are the elements of the identity matrix. One can simplify the right hand side of Eq. (A.1) by substituting Eq. (A.2) into Eq. (A.1) and performing the summation over j. Also the left hand side can be expressed as the product of the identity matrix times the matrix of partial derivatives of x_j^*. The result is

$$\sum_j \delta_{ij} \frac{\partial x_j^*}{\partial w_{rs}} = g_i'(u_i^*) \left\{ \delta_{ir} x_s^* + \sum_j w_{ij} \frac{\partial x_j^*}{\partial w_{rs}} \right\}. \tag{A.3}$$

On collecting all the partial derivatives in (A.3) on the left hand side, one obtains

$$\sum_k L_{ij} \frac{\partial x_j^*}{\partial w_{rs}} = \delta_{ir} g_i'(u_i^*) x_s^*, \tag{A.4}$$

where L_{ij} is given by

$$L_{ij} = \delta_{ij} - g_i'(u_i^*) w_{ij}. \tag{A.5}$$

Finally, multiply both sides of (A.4) by $(\mathbf{L}^{-1})_{ki}$, i.e., by the matrix elements of the inverse of $\mathbf{L}$, and sum over i. The result is

$$\frac{\partial x_k^*}{\partial w_{rs}} = (\mathbf{L}^{-1})_{kr} g_r'(u_r^*) x_s^*, \tag{A.6}$$

This is the desired result.

Appendix B

Almeida (1987) has shown that the convergence of the forward propagation implies the local stability of the backward propagation. To see why this must be the case it suffices to linearize Eq. (4.1) or Eq. (5.1) about a point attractor, i.e., $\mathbf{x} = \mathbf{x}^* + \boldsymbol{\epsilon}$. The resulting linear equation has the form (expressed in vector notation)

$$d\boldsymbol{\epsilon}/dt = -\mathbf{L}\boldsymbol{\epsilon}, \tag{B.1}$$

where $\mathbf{L}$ is given by Eq. (5.12). Now observe that the backward propagation Eq. (5.17) has the form

$$d\mathbf{y}/dt = -\mathbf{L}^T\mathbf{y} + \mathbf{b}, \tag{B.2}$$

where $b_k = g_k'(u_k)J_k$. Now, from (B.1) it is clear that the local stability of the forward equations depends on the eigenvalues of the matrix $\mathbf{L}$ and from Eq. (B.2) it is clear that the local stability of backward propagation

equations depends on the eigenvalues of the transposed matrix $\mathbf{L}^T$. But, $\mathbf{L}$ and its transpose $\mathbf{L}^T$ both have the same eigenvalues, hence if a fixed point of Eq. (5.12) is stable so is the corresponding fixed point of Eq. (5.17). A similar result holds for higher order neural network models.

Appendix C

This appendix discusses several issues relating to the implementation of the recurrent backpropagation algorithms on digital computers. The equations of motion can be solved with the usual numerical techniques for integrating differential equations, e.g., Euler or Runge-Kutta. In practice it turns out that the Euler method (also called first-order finite difference) suffices for converging onto the steady state solution. With a time step of $\Delta t = 1$ the forward propagation equations reduce to the forms used by Almeida (1987), Rohwer and Forrest (1987), Parker (1982), and others. These finite difference equations can exhibit oscillations which do not occur in the differential equations or in finite difference simulations with $\Delta t < 1$. This investigator typically uses $\Delta t = 0.9$ to suppress these oscillations.

A full dynamical simulation requires that the forward equations, the backward equations, and the weight update equations be solved simultaneously and that the patterns be presented randomly. In practice this integration is prohibitively time consuming. The usual approach is to integrate the forward propagation equations until they converge, then to integrate the backward propagation equations until they converge, and then to calculate the gradient. This is repeated for all the patterns while accumulating the gradient. The accumulation over all patterns makes the learning dynamics deterministic rather than stochastic and guarantees that the system will converge. However, it may only converge to a local minimum.

The mathematical form of the associative memory learning equation leads to certain computational efficiencies. First, it is not necessary to relax either the forward or backward propagation equations for the visible units because the fixed points of (8.12) can be calculated analytically for the visible units. Second, if there are no connections between hidden units it is not necessary to relax any of the equations in the network. The fixed points of the forward and backward propagation equations may be calculated in two iterations each, as if the system were a conventional feedforward network with a single hidden layer. Finally, if there are no hidden units the fixed points of the backward propagation equation can be calculated analytically without any iterations. This final case is the learning algorithm used by Samad and Harper (1987) in an associative memory which used an annealing scheme to recall memories rather than Eq. (5.1).

In networks with units with widely disparate fan-ins it is useful to multiply the initial weights by the inverse fan-in. If this is done, the gradient in the learning equations should be multiplied by the same factor. This suppresses saturation of the nodes with large fan-in and leads to improved performance of the learning algorithm.

A useful measure of progress during the learning process is the normalized error, E_n. This is defined to be

$$E_n = \frac{E}{E_{mean}}, \tag{C.1}$$

where E is the error function summed over all patterns and E_{mean} is

$$E_{mean} = \frac{1}{2} \sum_{\alpha} \sum_{i} |\eta_i^{\alpha} - \langle \eta_i \rangle|^2, \tag{C.2}$$

where η_i^{α} is the αth target output for the ith unit and where $\langle \eta_i \rangle$ is the average over patterns of the target values at the ith unit. E_n is useful because, independent of network topology and problem domain, the backpropagation algorithm learns the average output pattern very rapidly. Therefore the network initially goes to $E_n = 1$. As the distinctions between the patterns are learned, E_n gradually drops below one.

Acknowledgments

The author thanks Liam Healy for considerable assistance during the production of this paper and for providing crucial insight into the behavior of dissipative systems. Thanks also to Robert Jenkins, Juan Pineda, W. Jack Rugh, Terry Sejnowski, Kim Strohbehn, and Ben Yuhas for productive discussions. The two referees also contributed very constructive comments. This work was supported in part by the Air Force Office of Scientific Research under Grant AFOSR-87-0354 and by the Applied Physics Laboratory under IRAD-X8U.

References

Almeida, L. B. (1987), A learning rule for asynchronous perceptrons with feedback in a combinatorial environment, *in* "Proceedings of the IEEE First Annual International Conference on Neural Networks" (M. Caudil and C. Butler, Eds.), pp. 609–618, San Diego, CA.

Amari, Shun-Ichi (1972), Characteristics of random nets of analog neuron-like elements, *IEEE Trans. Syst. Man Cybernetics* 2, 643–657

Amari, Shun-Ichi (1977), A mathematical approach to neural systems, *in* "Systems Neuroscience" (J. Metzler, Ed.), pp. 67–118, Academic Press, New York.

Arnold, V. I. (1986), "Catastrophe Theory," Springer-Verlag, Berlin.

ATIYA, A. (1987), Learning on a general network, *in* "Proceedings of IEEE Conference on Neural Information Processing Systems" (D. Z. Anderson, ed.), Denver, CO, Nov. 8–12, to appear.

BALLARD, D. H. (1987), "Modular Learning in Neural Networks," AAAI Proceedings of 6th National Conference on Artificial Intelligence, pp. 279–284.

BAUM, E. B., AND WILCZEK, F. (1987), Supervised learning of probability distributions by neural networks, preprint.

BINDER, K., AND YOUNG, A. P. (1986), Spin glasses: Experimental facts, theoretical concepts, and open questions, *Rev. Mod. Phys.* **58**, 801–976.

COHEN, M. A., AND GROSSBERG, S. (1983), Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. IEEE Transactions on Systems, Man, and Cybernetics, SMC-13, pp 815–826.

FUKUSHIMA, K. (1987), A neural network model for selective attention in visual pattern recognition and associative recall, *Appl. Opt.* **26**(23), 4985–4992.

GEMAN, S. (1981), The law of large numbers in neural modelling, *in* "SIAM–AMS Proceedings," Vol. 13, pp. 91–105.

HINTON, G. E., SEJNOWSKI, T. J., AND ACKLEY, D. H. (1984), "Boltzmann Machines: Constraint Satisfaction Networks That Learn, Tech. Rep. No. CMU-CS-84-119, Carnegie-Mellon University, Dept. of Computer Science, Pittsburgh, PA.

HOPFIELD, J. J. (1982), Neural networks as physical systems with emergent collective computational abilities, *Proc. Natl. Acad. Sci. USA Bio.* **79**, 2554–2558.

HOPFIELD, J. J. (1984), Neurons with graded response have collective computational properties like those of two-state neurons, *Proc. Natl. Acad. Sci. USA Bio.* **81**, 3088–3092.

KOHONEN, T. (1984), "Self-Organization and Associative Memory," Springer-Verlag, Berlin.

LAPEDES, A., AND FARBER, R. (1986a), A self-optimizing, nonsymmetrical neural net for content addressable memory and pattern recognition, *Physica D* **22**, 247–259.

LAPEDES, A., AND FARBER, R. (1986b), Programming a massively parallel, computation universal system: Static behavior, *in* "Neural Networks for Computing" (J. S. Denker, Ed.), AIP Conference Proceedings, Vol. 151, pp. 283–298, Snowbird, UT.

LE CUN, Y. (1985), Une procedure d'appresentissage pour reseau a seuil assymetrique [A learning procedure for an asymmetric threshold network], *Proc. Cognitiva* **85**, 599–604.

MINSKY, M., AND PAPERT, S. (1969), "Perceptrons," MIT Press, Cambridge.

PARKER, D. B. (1982), "Learning-Logic," Invention Report, S81-64, File 1, Office of Technology Licensing, Stanford University.

PARKER, D. B. (1967), Second order backpropagation: Implementing an optimal $O(n)$ approximation to Newton's method as an artificial neural network, draft preprint obtained from author.

PINEDA, F. J. (1987a), Generalization of backpropagation to recurrent neural networks, *Phys. Rev. Lett.* **18**, 2229–2232.

PINEDA, F. J. (1987b), Generalization of backpropagation to recurrent and higher order networks, *in* "Proceedings of IEEE Conference on Neural Information Processing Systems" (D. Z. Anderson, Ed.), Denver, CO, Nov. 8–12, to appear.

ROHWER, R., AND FORREST, B. (1987), Training time dependence in neural networks, *in* "Proceedings of the IEEE First Annual International Conference on Neural Networks" (M. Caudil and C. Butler, Eds.), Vol. 2, pp. 701–708, San Diego, CA.

RUMELHART, D. E., HINTON, G. E., AND WILLIAMS, R. J. (1986), Learning internal representations by error propagation, *in* "Parallel Distributed Processing" (D. E. Rumelhart and J. L. McClelland, Eds.), pp 318–362, MIT Press, Cambridge.

SAMAD, T., AND HARPER, P. (1987), Associative memory storage using a variant of the generalized delta rule, *in* "Proceedings of the IEEE First Annual International Conference on Neural Networks" (M. Caudil and C. Butler, Eds.), Vol. 3, pp. 173–183, San Diego, CA.

SEJNOWSKI, T. J. (1981), Skeleton filters in the brain, *in* "Parallel Models of Associative Memory" (G. E. Hinton and J. A. Anderson, Eds.), pp. 189–212, Erlbaum, Hillsdale, NJ.

SIMARD, P. (1988), Private communication.

WHITE, H. (1987), Some asymptotic results for back-propagation, *in* "Proceedings of the IEEE First Annual International Conference on Neural Networks" (M. Caudil and C. Butler, Eds.), Vol. 3, pp. 261–266, San Diego, CA.

Submitted to news and views section of "Neural Computation"

Recurrent back-propagation and the dynamical approach to adaptive neural computation

Fernando J. Pineda

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, CA 91109

March 13, 1989

Abstract

"back-propagation" is the name given to a family of numerical techiques and adaptive models which have had a significant impact on neural computation and optimization. The classical numerical algorithm applies to discrete feedforward networks only. The extension of these ideas to recurrent networks leads naturally to a continuous-time formalism which may map onto collective physical systems.

The characteristic features of the formalism are presented without going too deeply into obscure mathematical details. The distinctions between the physical approach and the algorithmic approach are emphasized.

Recent developments in learning time-dependent states are discussed and finally, physical and biological plausibility concerns are discussed.

Introduction

The problem of loading neural networks with a nonlinear map can be likened to the problem of finding the parameters in a multidimensional nonlinear curve fit. The traditional way of accomplishing this is to minimize a measure of the error between the actual output and the "target" output. Many useful techniques exist, but the most common methods are methods which make use of gradient information. In general, if there are N free parameters in the objective function, the number of operations required to calculate the gradient numerically, is at best proportional to N^2. Neural networks are special because their mathematical form permits two tricks (to be discussed below) which reduce the complexity of the gradient calculation. When these two tricks are implemented, the gradient calculation scales linearly with the number of parameters (weights), rather than quadratically. The resulting algorithm is known as a back-propagation algorithm.

Classical back-propagation was introduced to the neural network community by Rumelhart, Hinton and Williams (1986). Essentially the same algorithm was developed independently by Werbos (1974) and Parker (1982) in different contexts. le Cun (1988) has provided a brief overview of back-propagation pre-history and stresses that the independent discovery of the technique and its interpretation in the context of connectionist systems is a recent and important development. He points out that within the framework of optimal control the essential features of the algorithm were known as early as 1969 (Bryson and Ho, 1969).

In this paper, the term "back-propagation" will be used generically to refer to any algorithm or dynamical system which calculates the gradient by exploiting the two tricks. Furthermore, since one can write a back-propagation routine for evaluating the gradient and then use this routine in any prepackaged numerical optimization package, it is reasonable to take the position that the term "back-propagation" should be attached to the way the gradient is calculated rather than to the particular algorithm for using the gradient (e.g. conjugate gradient, line search, etc.).

If neural networks were merely clever numerical algorithms it would be difficult to completely account for the frenzy now associated with the field. To my mind, much of the excitement is due to the work of Hopfield (1982) who made explicit the profound relationship between information storage and dynamically stable configurations of collective physical systems. Hopfield nets consist of interacting spins which together form a system known as a spin glass. Spin glasses are the classic example of a collective physical system. The relevent physical property of spin glasses which make them useful for computation is that the collective interactions between all the spins can result in stable patterns which can be identified with stored memories. Although it may not be particularly useful for practical computing, the Hopfield net serves as an explicit example of the principle of collective computation. Digital computers, on the other hand, can compute because they

are physical realizations of finite state machines. In digital computers collective dynamics does not play a role at the algorithm level, although it certainly plays a role at the implementation level since the physics of transistors is collective physics. Collective computation is the idea that collective dynamics plays an important role at the algorithm level as well as at the implementation level. The observation that collective dynamics can play a role at both levels suggests that an efficient approach would be to use the same collective dynamics at both levels. This is what one might call a physical approach to computation. Therefore, rather than have machine independent algorithms, one would have just the opposite extreme, in which the implementation medium would necessarily influence the design of algorithms. The physical approach constrains neural network models to be plausible as collective physical dynamical systems. The resulting "dynamical algorithms" could then fully exploit the collective behavior of physical hardware.

Recurrent back-propagation (RBP) is a non-algorithmic continuous-time formalism for adaptive recurrent and nonrecurrent networks in which the physical aspects of the computation are stressed (Pineda,1987a, 1987b,1988). The formalism is expressed in the language of differential equations so that the connection to collective physical systems is more natural. RBP can be put into an algorithmic form to optimize the performance of the network on digital machines, nevertheless, as shall be discussed below, the intent of the formalism is to stay as close to collective physics and dynamics as possible.

RBP has proven to be a rich and useful computational tool. Qian and Sejnowski (1988) have demonstrated that a recurrent back-propagation network can be trained to calculate stereo disparity. This results in a network similar to that of Marr and Poggio (1976). Barhen et al. (1989a) have used the method to train networks on inverse kinematics for robotic applications. The formalism has also been fertile soil for theoretical developements. Pearlmutter (1988) has extended the technique to time dependent trajectories while Simard and Ballard (1988) have investigated its convergence properties.

Overview of the Formalism

The class of neural network models which can be trained by RBP is very general, but it is useful to pick a definite system as an example, therefore consider a neural network model based on differential equations of the form

$$\tau_x dx_i/dt = -x_i + \sum_j w_{ij} f(x_j) + I_i \qquad (1)$$

The vector **x** represents the state vector of the network, **I** represents an external input vector and **w** represents a matrix of coupling constants (weights) which represent the strengths of the interactions between the various neurons. The relaxation time scale is τ_x. By hypothesis, the vector valued function $f(x_i)$ is differentiable and chosen so as to give the system appropriate dynamical properties.

For example, biologically motivated choices are the logistic or hyperbolic tangent functions (Cowan, 1968). When the matrix w is symmetric this system corresponds to the Hopfield model with graded neurons (1984).

In general , the solutions of equation (1) exhibit oscillations, convergence onto isolated fixed points and chaos. For our purposes, convergence onto isolated fixed points is the desired behavior, because we use the value of the fixed point as the output of the system. When the network is loaded, the weights are adjusted so that the output of the network is the desired output.

There are several ways to guarantee convergence. One way is to impose structure on the connectivity of the networks , e.g. a lower triangular weight matrix or a symmetric weight matrix. Symmetry, although mathematically elegant, is quite stringent because it constrains microscopic connectivity by requiring pairs of neurons to be symmetrically connected. A less stringent constraint is that of Guez et al. (1988) who used Gersgorin's eigenvalue localization theorem to show that asymptotic stability can be obtained by imposing constraints on the row norm of the matrix

$$L_{ij} = \delta_{ij} - w_{ij} f'(x_j) \qquad (2)$$

where δ_{ij} are the elements of the identity matrix and $f'(x_j)$ is the derivative of $f(x_j)$.

If the feedforward, symmetry or Guez stabilty conditions are imposed as initial conditions on a network, gradient descent dynamics will typically converge onto a network which violates the conditions. Nevertheless, this author has never observed an initially stable network becoming unstable while undergoing simple gradient descent dynamics. This fact points out that the stability conditions are merely sufficient conditions -- they are not necessary. This fact also motivates the stability conjecture upon which recurrent back-propagation is based: that if the initial network is stable, then the gradient descent dynamics will not change the stability of the network. The reader should note the following caveat: that this conjecture is a statement about the kinds of problems attacked by this author rather than a statement about rigourous mathematics.

In gradient descent learning, the computational problem is to optimize an objective function whose free parameters are the weights. Let the number of weights be denoted by "N" and let the number of processing units be denoted by "n". Then, N is proportional to n^2 if the fan-in/fan-out of the units is proportional to n. In a neural network the evaluation of an objective function requires O(N) or $O(n^2)$ operations(1). Accordingly, to calculate the gradient of the objective function by numerical differentiation requires $O(N^2)$ or $O(n^4)$ calculations. For big problems, i.e. problems with lots of connections this becomes intractable very rapidly. This notion of big should not be confused with the difficulty of a problem in the sense of whether a problem is NP complete or not.

(1) The notation $O(n^2)$ means that in the large n limit the number of operations is bounded by Cn^2 where C is a constant.

Furthermore, the scaling referred to here should not be confused with the number of gradient evaluations required for convergence to a solution. Indeed, for some problems, e.g. parity, the required number of gradient evaluations may diverge at critical training set sizes (Tesauro, 1987).

Now, as already mentioned, back-propagation adaptive dynamics is based on gradient descent and exploits two tricks to reduce the amount of computation. The first trick uses the fact that, for equations of the form (1), the gradient of an objective function $E(\mathbf{x}^o)$ can be written as an outer-product, i.e.

$$\nabla_{\mathbf{w}} E = \mathbf{y}^o \mathbf{f}(\mathbf{x}^o)^T \quad . \tag{3}$$

Where $\mathbf{x}^o$ is the fixed point of eqn. (1) and where the "error vector" $\mathbf{y}^o$ is given by

$$\mathbf{y}^o = (\mathbf{L}^T)^{-1}\mathbf{J} \tag{4}$$

where $\mathbf{L}^T$ is the transpose of the matrix defined eqn. (2) n x n matrix and $\mathbf{J}$ is an external error signal which depends on the objective function and on $\mathbf{x}^o$. This trick reduces the computational complexity of the gradient calculation by a factor of n because $\mathbf{L}^{-1}$ can be calculated by direct matrix inversion in $O(n^3)$ operations and because $\mathbf{x}^o$ can be calculated in only $O(n^2)$ calculations. Thus the entire calculation scales like $O(n^3)$ or $O(N^{3/2})$. The second trick exploits the fact that $\mathbf{y}^o$ can be calculated by relaxation or equivalently it is the (stable) fixed point of the following couple set of linear differential equation:

$$\tau_y dy_i/dt = -y_i + f'(x_i)\sum_j w_{ji} y_j + J_i \tag{5}$$

A form of this equation was derived by Pineda, (1987a). A discrete-time version was derived independently by Almeida (1987). To relax y (i.e. to integrate eqn. (5) until y reaches steady state) requires $O(n^2)$ operations per time step. The number of time steps is independent of n. Therefore the calculation of $\mathbf{y}^o$ is $O(n^2)$ or O(N). The method is computationally efficient provided the network is sufficiently large and sparse and provided that the fixed points are not marginally stable. These results are summarized in table 1. Note that the two back-propagation algorithms scale like O(N), but this hides the constants of proportionality which for feed-forward networks depends on the number of layers where as for recurrent networks the constant of proportionality depends strongly on the eigenvalues of the $\mathbf{L}$ matrix. Indeed, if the fixed points are marginally stable, the number of iterations required to converge onto $\mathbf{x}^o$ and $\mathbf{y}^o$ may diverge.

numerical algorithm	**complexity**
Worst case (e.g. numerical differentiation)	$O(N^2)$
matrix inversion (e.g. gaussian elimination)	$O(N^{3/2})$
matrix inversion by relaxation (e.g. RBP)	O(N)
recursion (e.g. classical feed-forward back-propagation)	O(N)

Table 1. Scaling of various algorithms with the number of connections

For all its faults, back-propagation has permitted optimization to be applied to problems which were previously considered numerically intractable. The O(N) scaling of the gradient calculation is arguably the single most important reason that back-propagation algorithms have made such an impact. The idea of using gradient descent is certainly not new, but whereas it was previously tractable on small problems only, it is now tractable on big problems to. It is interesting to observe that a similar situation arose after the development of the FFT algorithm. The idea of numerical fourier transforms had been around for a long time before the FFT, but the FFT caused a computational revolution by reducing the complexity of an n-point fourier transform, from $O(n^2)$ to $O(n \cdot \log(n))$.

Dynamical vs Algorithmic approaches

Back-propagation algorithms are usually viewed from an algorithmic viewpoint. For example, the gradient descent version of the algorithm is expressed in the following pseudo-code:

```
while(E > ε)
{
    initialize weight change Δw = 0
    repeat for each pattern
    {
        relax eqn. (1) to obtain x^o
        relax eqn. (4) to obtain y^o
        calculate gradient ∇E = y^o f(x^o)^T
        accumulate gradients Δw = Δw + ∇E
    }
    update weights w = w + Δw
}
```

Note that all the patterns are presented before a weight update. On the other hand, a "dynamical algorithm" can be obtained by replacing the weight update step with a differential equation, i.e.

$$\tau_w dw_{ij}/dt = y_i f(x_j). \qquad (6)$$

and integrating it simultaneously with the forward-propagation and backward-propagation equations. A constant pattern is presented through the input pattern vector **I** and the error signal is presented through the error vector **J**. The dynamics of this system is capable of learning a single pattern so long as the relaxation time of the forward and backward propagations (τ_x and τ_y) is much slower than the relaxation time of the weights, τ_w . Since the forward and backward equations settle rapidly after a presentation, the outer product $\mathbf{y}\mathbf{f}(\mathbf{x})^T$ is a very good approximation for the gradient during most of the integration. To learn multiple patterns, the patterns must be switched slowly compared to the settling time of the forward and backward equations, but rapidly compared to τ_w, the time scale over which the weights change.

The conceptual advantage of this approach is that one now has a dynamical system which can be studied and perhaps used as a basis for models of actual physical or biological systems. This is not to say that merely converting an algorithm into a dynamical form makes it biologically or physically plausible. It simply provides a starting point for further development and investigation.

Intuition and formal results concerning algorithmic models do not necessarily apply to the corresponding dynamical models. For example, consider the well known "fact" that gradient descent is a poor algorithm compared to conjugate gradient. In fact this conventional wisdom is incorrect when it comes to physical dynamical systems. The reason is that the disease which makes gradient descent inefficient is a consequence of discretization. The difficulty occurs when descending down a long narrow valley. Gradient descent can wind up taking many tiny steps crossing and re-crossing the actual gradient direction. This is inefficient because the gradient must be recomputed for each step and because the amount of computation required to recalculate the gradient from one step to the next is approximately constant. Conjugate gradient is a technique which assures that the new direction is conjugate to the previous direction and therefore avoids the problem. Accordingly larger steps may be taken and less gradient evaluations are required.

On the other hand gradient descent is quite satisfactory in physical dynamical systems simply because time is continuous. The "steps" are by definition infinitely small and the gradient is evaluated continuously. No repeated crossing of the gradient direction occurs. For the same reason, the ultimate performance of physical neural networks cannot be determined from how quickly or how slowly a "neural" simulation runs on a digital machine. Instead one must integrate the simultaneous equations and measure how long it takes to learn, in multiples of the fundamental time scales of the equations. As an example, consider the following illustrative problem. Chose input and output vectors to be randomly selected 5 digit binary vectors scaled between 0.1 and 0.9. Use a network with two layers of five units each with connections going in both directions (50 weights). For dimensionless time scales chose $\tau_x = \tau_y = 1.0$, $\tau_w = 32\ \tau_x$ and select a new pattern at random every $4\tau_x$. The equations may be integrated crudely, e.g. use the Euler method with ($\Delta t = 0.02\ \tau_x$). One finds that the error reaches E = 0.1 in approximately $4 \times 10^3\ \tau_x$ or after 10^3 presentations. Figure 1. shows the error as a function of time.

// INSERT FIG. 1 HERE //

To estimate the performance of an electronic physical system we can replace these time scales with electronic time scales. Therefore, suppose patterns are presented every 10^{-5}sec (100 kHz). This is the performance bottleneck of the system, since the relaxation time of the circuit, τ_x is then approximately 2.5×10^{-6}sec, which is slow compared with what can be achieved in analog VLSI. Hence in this case the patterns would be learned in approximately 10 milliseconds.

Unlike simple feedforward networks, recurrent networks exhibit dynamical phenomena. For example, a peculiar phenomenon can occur if a recurrent network is trained as an associative memory to store multiple memories: it is found that the objective function can be reduced to some very small value, yet when the network is tested for recall, the supposedly stored memory is missing! This is due to a fundamental limitation of gradient descent. Gradient descent is capable of moving existing fixed points only. It cannot create new fixed points. To create new fixed points requires a technique whereby some degrees of freedom in the network are clamped during the loading phase and released during recall phase. The analogous technique in feed-forward networks is called teacher forcing. It can be shown that this technique causes the creation of new fixed points. Unfortunately, after the suppressed degrees of freedom are released, there is no guarantee that the system is stable with respect to the suppressed degrees of freedom. Therefore the fixed points sometimes turn out to be repellors instead of attractors. In feed-forward nets teacher forcing causes no such difficulties because there is no dynamics in feed-forward networks and hence no attractors or repellors.

Recent Developments

Zak, (1988) has suggested the use of fixed points with infinite stability in recurrent networks. These fixed points, denoted "terminal attractors", have two properties which follow from their infinite stability. First, their stability is always gauranteed, hence the repellor problem never occurs, and second trajectories converge onto them in a finite amount of time, rather than an infinite amount of time. In particular, if a terminal attractor is used in the weight update equation, a remarkable speedup in learning time occurs, see e.g. Barhen (1989a). These interesting properties are a consequence of the fact that the attractors violate the Lipschitz condition.

Pearlmutter, (1989) has extended the recurrent formalism to include time-dependent trajectories (time-dependent recurrent back-propagation or TDRBP). In this approach the objective function of the fixed point is replaced with an objective functional of the trajectory. The technique is the continous time generalization of the sequence generating network discussed by Rumelhart et al. (1986). Like all back-propagation algorithms the amount of calculation scales like O(N) for each gradient evaluation. However, like the Rumelhart network, it requires that the network be unfolded in time during training. Hence the storage during training scales like O(mN) where m is the number of unfolded time steps. Furthermore, the technique is acausal in that the back-propagation equation must be integrated backwards in time. This merely reflects the fact that one is solving a two-point boundary problem of the kind familiar from control theory. For problems where the target trajectories are known apriori and on-line learning is not required, this is the technique of choice.

On the other hand a causal algorithm has been suggested by Williams and Zipser (1989). This algorithm does not take advantage of the back-propagation tricks and therefore the complexity scales like $O(N^2)$ for each gradient evaluation. Nevertheless, for small problems where on-line learning is required it is the technique of choice.

Both techniques seek to minimize a measure of the error between a target trajectory and an actual trajectory by performing gradient descent. Only the method used for the gradient evaluatinon differs. Therefore one expects that, to the extent that on-line training is not an issue and to the extent that complexity is not an issue, one could use the two techniques interchangably to create networks.

Both techniques can suffer from the repellor problem if an attempt is made to introduce multiple attractors. As before, this problem could be solved by introducing a time dependent terminal attractor.

Discussion

Biologically and physically plausible adaptive systems should satisfy certain constraints. 1) They should scale well with connectivity, e.g. linearly 2) they should require no global synchronization 3) they should use low precision components and 4) they should not impose unreasonable structural contraints, e.g. symmetric weights or bi-directional signal propagation. Back-propagation algorithms in general and RBP and TDRBP in particular can be considered in the light of each of these constraints.

Linear scaling of the gradient calculation in back-propagaton algorithms is a consequence of the local nature of the computation, i.e. that each unit only requires information from the units to which it is connected. This notion of locality, which arises from the analysis of the numerical algorithm is distinct from the notion of spatial locality, which is a constraint imposed by physical space on physical networks. Spatial locality is how one avoids the $O(n^2)$ growth of wires in networks. Both locality constraints could be satisfied by physical back-propagation networks.

Global synchronization requires global connections, therefore it is undesirable if the network is to scale up. In one sense, the problem of synchronization has been eliminated in RBP because there is no longer any need for separate forward , backward and update steps, indeed equations (1), (5) and (6) are "integrated" simultaneously by the dynamical system as it evolves. There is another sense in which synchronization causes difficulties. In physical systems and in massively parallel digital simulations,time delays and asynchronous updates, can give rise to chaotic or exponential stochastic behavior (Barhen, 1989b). Barhen et al. have shown that this "emergent chaos" can be suppressed easily by the appropriate choice of dynamical parameters.

It is still an open question as to whether back-propagation algorithms require low precision or high precision components. Formal results suggest that some problems, like parity in single layer nets (Minsky, 1969), may lead to exponential growth of weights. In practice it appears that 16 bits of precision for the weights and 8 bits of precision for the activations and error signals are sufficient for many useful problems (Durbin, 1987).

Structurally, RBP and TDRBP impose no constraints on the weight matrix. Furthermore, in RBP networks there appears to be no need to take special measures to insure the stability of the network while undergoing training. This would help the biological plausiblity of the model were it not for the requirement that the connections be bi-directional. Bi-directionality is arguably the biggest plausibility problem with the algorithms based on backpropagation. Biologically, this requires bi-directional synapses or separate, but equal and opposite, paths for error and activation signals. There is no evidence for either structure in biological systems. The same difficulties arise in electronic implementations where engineering solutions to this problem have been developed (Furman and Abidi,1988), but one would hope that a better adaptive dynamics would eliminate the problem altogether.

Acknowledgements

The author wishes to acknowledge very helpful discussions with : Pierre Baldi, Richard Durbin and Terrence Sejnowski. The author's research was funded, in part, by the Air Force Office of Scientific Research (AFOSR-87-0354) and was conducted while the author was employed by the Applied Physics Laboratory, The Johns Hopkins University.

References

Almeida, L. B. (1987). A Learning rule for asynchronous perceptrons with feedback in a combinatorial environment. In: *Proceedings of the IEEE First International Converence on Neural Networks*, (eds. M. Caudil and C. Butler), Vol. 2., pp. 609-618, San Diego CA

Barhen, J., Gulati, S., Zak, M. (1989), Neural Learning of Inverse Kinematics for Redundant Manipulators in Unstructured Environments, To appear in: *IEEE Computer*, June 1989, Special issue on Autonomous Intelligent Machines.

Barhen, J. and Gulati, S. (1989). "Chaotic relaxation" in concurrently asynchronous neurodynamics. Submitted to: *1989 International Joint Conference on Neural Networks*, June 18-19, Washington D.C.

Bryson, A.E. Jr., and Ho, Yu-Chi (1969). "Applied Optimal Control", Blaisdell Publishing Co.

Cowan, J. D. (1968). Statistical mechanics of nervous nets. In: *Neural Networks* (ed. E. R. Caianiello), pp. 181-188, Berlin, Springer-Verlag.

Durbin, Richard (1987). *Back-propagation with integers*, Neural Networks for Computing, Snowbird UT, 1987, (Abstracts of the meeting)

Furman, B. and Abidi, A. (1988). A CMOS backward error propagation LSI. *Proc. 22nd Asilomar Conf. on SIgnals, Systems and Computers*. Pacific Grove, CA, Nov. 1988

Guez,A, Protopopsecu, V. and Barhen, J. (1988).On the Stability, Storage Capacity , and Design of Nonlinear Continuous Neural Networks. *IEEE Trans. SMC* 18, pp.80-87

Hopfield, J. J. (1982). Neural Networks as Physical Systems with Emergent Collective Computational Abilities. *Proc. Nat. Acad. Sci. USA, Bio.* 79, pp. 2554-2558

Hopfield, J. J. (1984). Neurons with graded response have collective computational properties like those of two-state neurons. *Proc. Nat. Acad. Sci. USA, Bio.* 81, pp. 3088-3092

le Cun, Yann (1988). A theoretical Framework for Back-Propagation, *Proceedings of the 1988 connectionist models summer school, Carnegie-Mellon University*, (eds. D. Touretzky, G. Hinton, T. Sejnowski), Morgan Kaufmann, 1989.

Marr, D. and Poggio, T. (1976). Cooperative Computation of Stereo Disparity, *Science*, 194, 283-287

Minsky, M. and Papert, S. (1988), "Perceptrons" 2nd edition, M.I.T. Press, Cambridge, MA

Parker, David B. (1982). Learning-Logic, Invention Report, S81-64, File 1, Office of Technology Licensing, Stanford University

Pearlmutter, Barak A. (1989). Learning State Space Trajectories in Recurrent Neural Networks, *Neural Computation*, (this issue)

Pineda, F. J. (1987a). Generalization of back-propagation to recurrent neural networks. *Phys. Rev. Lett.* 18, pp. 2229-2232

Pineda, F. J. (1987b). Generalization of back-propagation to recurrent and higher order networks. In: *Proceedings of IEEE Conference on Neural Information Processing Systems.* (ed. D. Z. Anderson), Denver Colorado, Nov. 8-12

Pineda, F. J. (1988). Dynamics and Architecture for Neural Computation. *Journal of Complexity.* 4, pp. 216-245

Qian, Ning and Sejnowski, Terrence J., (1988). Learning to Solve Random-Dot Sterograms of Dense Transparent Surfaces with Recurrent Backpropagation, ***

Simard, Patrice Y., Ottaway, Mary B. and Ballard, Dana H. (1988). Analysis of recurrent backpropagation, *Proceedings of the 1988 Connectionist Models Summer School,* June 17-26, 1988, Carnegie Mellon Univ., Morgan Kaufman Publishers

Tesauro, Gerald, (1987). Scaling relationships in back-propagation learning: dependence on training set size, *Complex Systems,* 1, pp. 367-372

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Internal Representations by Error Propagation. In: *Parallel Distributed Processing,* Vol. 1, (eds. D. E. Rumelhart and J. L. McClelland), pp. 318-362

Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the hehavioral sciences, Ph.D. thesis Harvard University

Willaims, Ronald J., Zipser, David (1989). A learning Algorithm for Continually Running Fully Recurrent Neural Networks, *Neural Computation,* (this issue)

Zak, Michail, (1969). Terminal Attractors for Addressable Memory in Neural Networks, *Physics Letters A,* 133, pp. 18-22

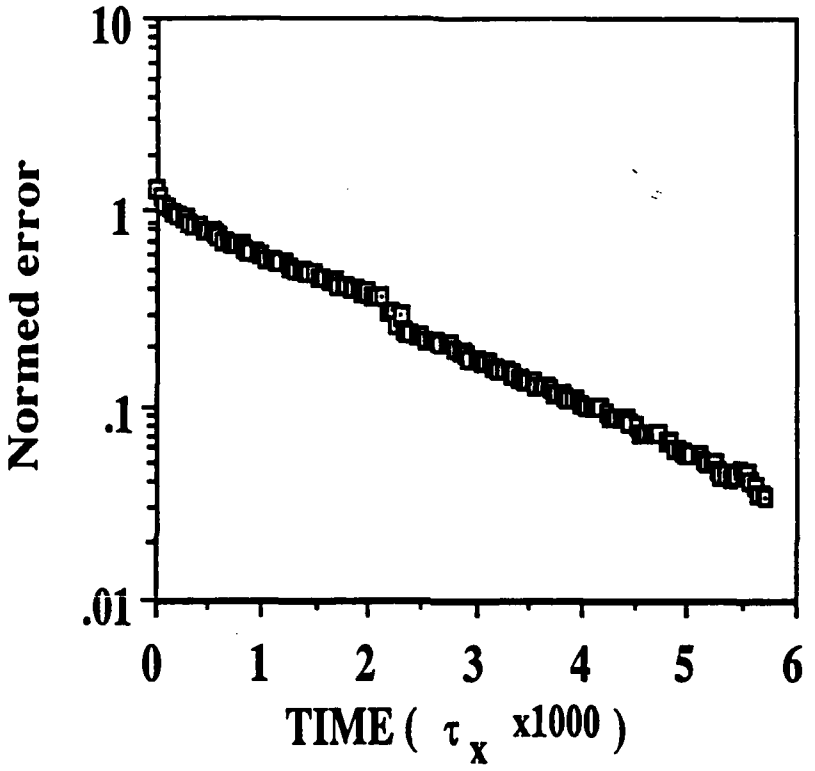

Figure 1. Error as a function of time